ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

"Desarrollo de una aplicación de Realidad Aumentada mediante la arquitectura Vuforia para la obtención de información de cuadros"

Alumno: César IñarreaSagüés

Tutor: IosuAzkue

Pamplona, 15 de Noviembre de 2012

Índice

0. Introducción

En los últimos años estamos viviendo el nacimiento a nivel público de una tecnología propia de las películas de ciencia ficción de cuando éramos niños y veíamos como aquel robot venido del futuro obtenía datos sobre personas con solo mirarles. Esa tecnología, mezcla entre la realidad virtual propia de los videojuegos o esas toscas máquinas de los salones de juego y la imagen captada por una cámara cualquiera de la realidad apuntada, es ni más ni menos la tan de moda “Realidad Aumentada” con la que empresas tan importantes como Google o Windows están empezando a sacar productos al mercado como gafas o aplicaciones móviles para Smartphones.

En el marco de este proyecto, la realidad aumentada se aplicará a cuadros famosos que mediante una aplicación Android nos proporciona información acerca de los mismos; para ello nos apoyamos en el API de Qualcomm (Vuforia) dirigido a la RA. Es por ello que en el desarrollo de la memoria no entraremos en profundidad en el funcionamiento de esta API sino que trataremos aquella parte creada y/o añadida por el autor del proyecto. Del mismo modo, solo se presentara aquella parte del código y funcionamiento de la aplicación diseñada y creada por el mismo.

1. Conceptos Generales

1. Realidad Aumentada

La realidad aumentada (RA) es la denominación que recibe una tecnología que nos permite complementar la percepción directa o indirecta de la realidad física con elementos virtuales generados por un ordenador, generando una realidad “mixta” entre ambas cosas.

La realidad aumentada está relacionada con la más popular Realidad Virtual y comparte con ésta algunas propiedades y características como son la generación de objetos y elementos 2D y 3D mediante ordenador y con el objeto de presentarlos al usuario en su campo visual. La diferencia radica en que mientras que la RV consiste en sustituir la realidad percibida por un entorno 100% generado, en la RA lo que se pretende es añadir y complementar el mundo real percibido por el sujeto con información adicional, el usuario nunca pierde el contacto con el mundo real y al mismo tiempo es capaz de recibir más información e incluso interactuar con ella.

Para conseguir añadir esos elementos adicionales a la visión percibida hemos de cumplir unas condiciones y hemos de poder contar con ciertos elementos necesarios para su realización. En el sentido más estricto de la Realidad Aumentada, como se puede observar en cazas en muchas películas (Fig. 1.1.1), solo precisamos de un soporte de representación de imágenes transparente que al mismo tiempo no permita ver lo que hay más allá de este. Pero en un concepto más amplio y veraz con lo que conocemos hoy como Realidad Aumentada, para la producción hemos de tener un dispositivo de captación de imágenes(cámara) para la captación del mundo real al que queremos añadir los elementos correspondientes, un dispositivo de representación de imágenes(por lo general una pantalla) que nos permita representar de forma conjunta tanto la realidad captada como los elementos virtuales que le asignamos y un dispositivo de computación que nos proporcione la capacidad de procesar las imágenes captadas por la cámara, analizar estas imágenes, reconocer los patrones predefinidos, generar los elementos que correspondan y por último enviar el conjunto a la pantalla (Fig. 1.1.2).

Fig. 1.1.1 RA en la cabina de un caza.

Fig. 1.1.2 RA en una Carta del Beisbol con representación 3D.

Además, podremos dotar a la realidad aumentada de elementos interactuables, como por ejemplo botones, que nos permitan comunicarnos con el programa y que reaccione realizando determinada acción como puede ser animar un objeto 3D, cambiar los colores de este o sirviendo como enlace a páginas web. Posteriormente daremos una explicación más técnica y detallada sobre el funcionamiento de la RA en nuestro proyecto.

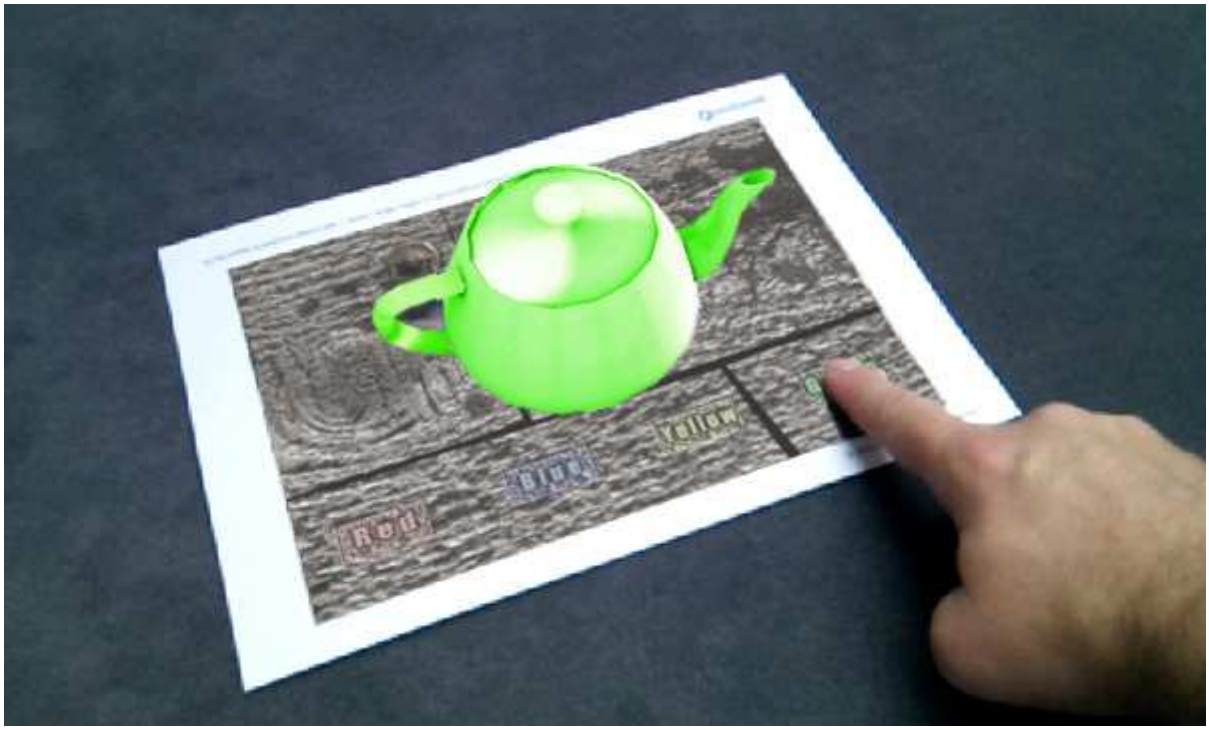


Fig. 1.1.3 RA con interacción con usuario.

Esta tecnología está introduciéndose en nuevas áreas de aplicación como son entre otras la reconstrucción del patrimonio histórico, el entrenamiento de operarios de procesos industriales, marketing, el mundo del diseño interiorista y guías de museos. El mundo académico no está al margen de estas iniciativas y también ha empezado a introducir la tecnología de la Realidad Aumentada en algunas de sus disciplinas. Sin embargo el conocimiento y la aplicabilidad de esta tecnología en la docencia es mínima; entre otros motivos se debe a la propia naturaleza, el costo de los elementos necesarios y estado de desarrollo de dicha tecnología, así como también a su escasa presencia en los ámbitos cotidianos de la sociedad, pese a que en los últimos tiempo esto está cambiando y podemos observar como poco a poco se está introduciendo en el dominio público. El desarrollo de iniciativas en la utilización de esta tecnología en la educación y su divulgación contribuirán a su extensión en la comunidad docente.

2. Smartphone

El término *Smartphone* es la denominación comercial con la que se refieren los fabricantes actuales a los móviles de última generación. El prefijo “smart", inteligente en inglés, hace referencia a la capacidad de los dispositivos, en este caso móviles, de conectividad, computación, cálculo, instalación de aplicaciones con diferentes fines, agenda digital... etc.

En definitiva, un *Smarphone* nos ofrece aquellas funcionalidades propias de un teléfono móvil tradicional tales como llamar o enviar SMS, a las que añade otras aplicaciones prácticas propias de las ya casi desaparecidas PDA’s o de ordenadores de sobremesa. Así, los *Smartphone* funcionan sobre sistemas operativos móviles

diseñados y pensados para el soporte de las funcionalidades básicas de un teléfono y al mismo tiempo proporcionando la capacidad a los fabricantes de estos, a operadoras móviles o a terceros, de construir sus propias herramientas software o aplicaciones para la plataforma específica con diferentes fines, desde el entretenimiento personal hasta el acceso a internet.

Gracias a su flexibilidad y comodidad de uso, la utilización de estos dispositivos, mitad teléfono móvil mitad ordenador, ha aumentado rápidamente en la última década sustituyendo a las ya mencionadas PDA en entornos profesionales tanto debido al aumento de prestaciones respecto a estas como a la posibilidad de acceder al correo allá donde estés; gracias a su vez a al 3G, otra de las ventajas proporcionadas por estos dispositivos. Al mismo tiempo, el crecimiento en el entorno público no ha sido menos y en especial entre los jóvenes, nuestro país en cabeza a nivel mundial, debido sobre todo a la aparición de aplicaciones de comunicación a tiempo real (chats) como el Whatsapp que permite estar en contacto entre la población de forma gratuita al tener una conexión de internet (wifi o 3g) contratada con algún operador. Este crecimiento se ve claro en la gráfica (Figura 1.2.1) donde podemos observar como el número de dispositivos se ha disparado en los últimos 5 años.

El crecimiento cualitativo ha crecido al mismo ritmo que el cuantitativo de estos dispositivos y cada año vemos cómo los usuarios exigen más memoria, más velocidad de los procesadores y demás factores de los terminales que adquieren hasta llegar al punto que muchas de las tareas que en un comienzo eran exclusivas de nuestros ordenadores personales, ahora son capaces de realizarse con nuestros teléfonos móviles.

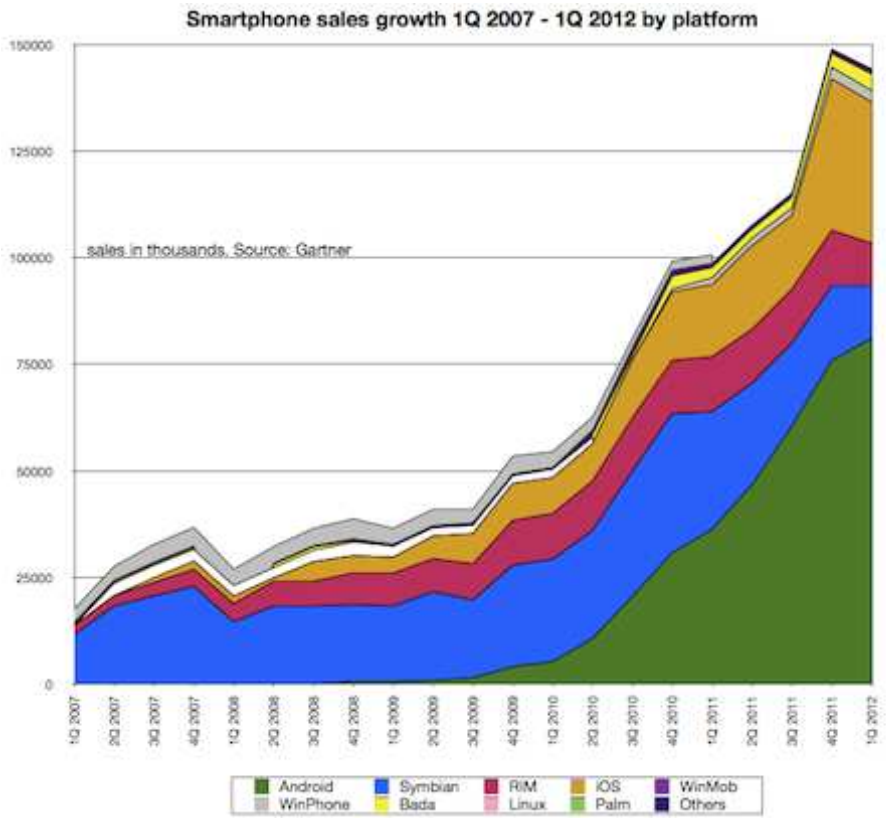


Fig. 1.2.1 Crecimiento del los Smartphones

3. Android

Android es un sistema operativo móvil basado en Linux, que junto a un conjunto de aplicaciones y herramientas middleware trata de proporcionar una plataforma de desarrollo para dispositivos como smartphones o tablets y en los últimos tiempos se está extendiendo a televisiones y otros dispositivos. Al igual que otros sistemas operativos móviles como iOS, Symbian o BlackBerry, fue pensado específicamente para smartphones pero sin embargo Android presenta una peculiaridad que le ha permitido crecer a un ritmo mucho mayor que sus competidores hasta llegar ser la plataforma más extendida. Esta peculiaridad es que al estar basado en Linux, resulta una plataforma libre, gratuita y multiplataforma.

Fig.1.3.1 Logotipo de Android

Otra característica que lo hace popular entre los desarrolladores de aplicaciones es el hecho de estar pensado en un sublenguaje de Java, Dalvik, que facilita el desarrollo de estas aplicaciones al ser un lenguaje muy extendido y conocido. Esto proporciona herramientas de acceso a componentes del teléfono o dispositivo tales como GPS o la agenda de una manera fácil e intuitiva. Además, Android proporciona una API (Application Programming Interface) que nos extiende la ya extensa API de Java pero más enfocada a la optimización con el propio sistema operativo.

Así pues, Android nos proporciona una arquitectura en 5 niveles posibilitando el desarrollo de las aplicaciones que interactúen con los diferentes componentes del dispositivo:

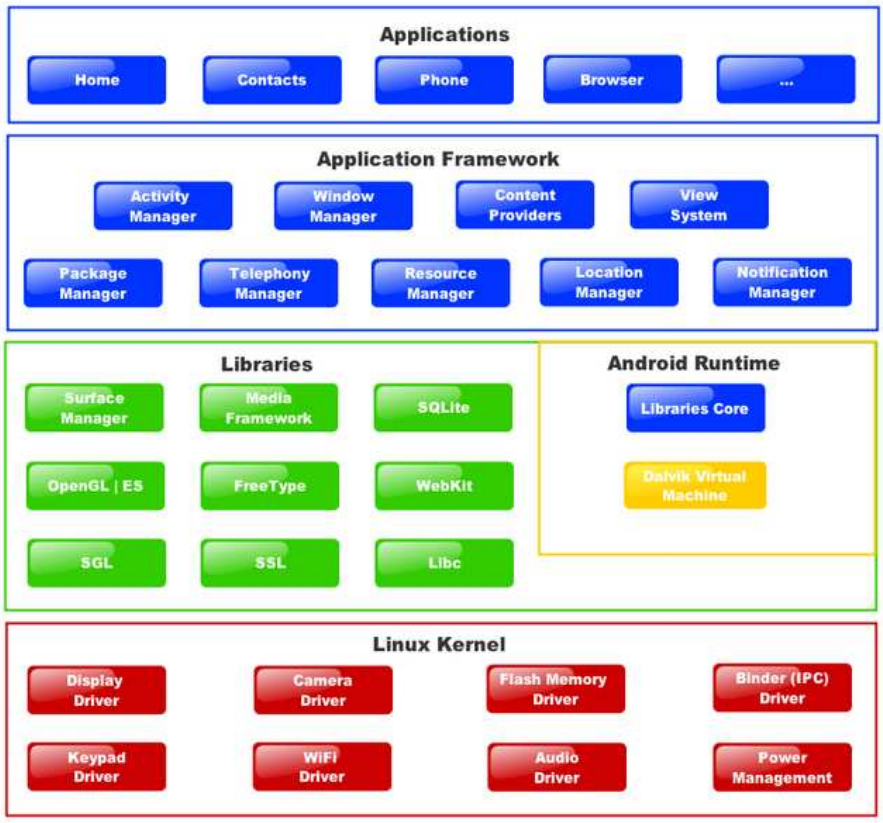


Fig.1.3.2 Arquitectura de Android

2. Análisis de requisitos

1. Objetivos Generales del Proyecto

Como ya hemos expuesto en la introducción del mismo, con el proyecto se pretende crear una aplicación de Realidad Aumentada que nos facilite y/o proporcione información acerca de un conjunto de cuadros famosos de forma intuitiva y visual utilizando para ello un Smartphone cualquiera con sistema operativo Android y no verse obligado a recurrir a pesadas búsquedas en diferentes medios, como puede ser internet, folletos de turismo o libros especializados. Sin embargo, la calidad y veracidad de la información suministrada mediante nuestra aplicación corresponderá a una futura línea de trabajo a desarrollar por profesionales del sector de la historia del arte.

2. Rendimiento requerido

En una primera parte de la aplicación, al tratarse de una aplicación de Realidad Aumentada, la aplicación tiene una fuerte dependencia sobre qué es lo que captamos en cada instante y por lo tanto se pretende que la recreación de los objetos añadidos al mundo real sea en tiempo real y sin cortes o retardos. Así pues, se ha trabajado en busca de la mayor eficiencia y eficacia de las funcionalidades de las que se ha proporcionado a la aplicación.

En una segunda parte, el funcionamiento y respuesta en tiempo real deja de ser crítico para nuestra aplicación y nos centramos en mayor medida en la presentación de forma clara de la información suministrada.

3. Definición de requisitos

Los requisitos del proyecto podemos diferenciarlos entre aquellos que hacen referencia al material o software necesario para la creación o desarrollo del proyecto, los que hacen referencia que se precisa para su funcionamiento o por último como ha de ser este (aspecto visual y funcionalidades).

1. Requisitos de Desarrollo

Ordenador

Para el desarrollo de nuestro proyecto precisamos de un ordenador en el que programar la aplicación, crear tanto los modelos reconocibles como las imágenes que se representarán al ser reconocidos y al no disponer de los cuadros originales, utilizamos el ordenador para bajarnos una imagen del cuadro de internet y utilizarlo en vez del original.

Además también lo utilizamos para la redacción de esta misma memoria.

Smartphone

Como ya se ha explicado antes, el proyecto consistirá en una aplicación para estos dispositivos (también funcionará en tablets, pero en un principio no está pensado para estas), por lo tanto para el desarrollo y la comprobación del correcto funcionamiento de la misma, necesitaremos un Smartphone. Este Smartphone ha de contar a su vez con una cámara, característica prácticamente proporcionada por la totalidad de dispositivos del mercado actual.

En el desarrollo de la aplicación podríamos haber optado por utilizar un simulador de Android, facilitado junto al SDK, pero al precisar del acceso a una cámara que nos proporcione imágenes a tiempo real, esa opción no nos vale ya que ningún simulador de los que se tuvo acceso lo permite.

En nuestro caso, el alumno proporciona el suyo propio con el que se realizan todas las fases del desarrollo y la presentación del proyecto ante el tribunal.

Eclipse

Se trata de un IDE o Integrated Development Environment, es decir un entorno integrado de desarrollo para diferente lenguajes, como en este caso es Dalvik (Java) que nos facilita el trabajo proporcionando herramientas para comunicarnos con el móvil, detección de errores en el código, visualización del mismo en diferente colores para identificar partes del mismo..etc.

Todo ello más extensamente explicado en el próximo apartado, junto a las instrucciones de instalación

Herramientas de desarrollo en Android

Para el desarrollo de aplicaciones de Android necesitamos un conjunto de librerías y herramientas que nos permiten compilar el código generado por nosotros mismo y el proporcionado por Vuforia (Qualcomm).

En nuestro proyecto utilizamos 3 de estas herramientas: el SDK oficial de Android, el ADT y el NDK oficial. Ambos los explicaremos en el apartado siguiente con más detalle.

Cygwin

A pesar del hecho de estar trabajando en un entorno en el sistema operativo Windows, la compilación de los componentes de la aplicación se realiza siguiente comandos de Linux, como son el NDK y el makefile. Por ello, trabajamos con este programa que nos realiza el trabajo de "traducción" de las órdenes al sistema.

Nuevamente, esta herramienta se explica más adelante con profundidad extendida.

SDK Vuforia

Se trata de un *Software Development Kit*, es decir, un conjunto de librerías que nos permitan programar nuestra aplicación de acuerdo a una API desarrollada por Qualcomm con la denominación *Vuforia* para la creación y desarrollo de aplicaciones de Realidad Aumentada.

Fig. 2.3.1 Logo Vuforia

Como los anteriores requisitos de desarrollo, es en el siguiente apartado donde explicaremos extensamente en qué consiste y cómo funciona.

2. Requisitos para su Funcionamiento

Smartphone

Como ya sabemos, la aplicación correrá en un dispositivo de este tipo (aunque también sabemos que en una tablet con cámara también funcionaria). Por lo tanto para usar la aplicación resultante hemos de contar con un smartphone con cámara y una capacidad de procesamiento medio.

Es decir, nos basta con disponer de un smartphone de gama media para ejecutar nuestra aplicación.

Cuadro

Al tratarse de una aplicación de Realidad Aumentada sobre cuadros, lógicamente necesitamos disponer del mismo delante para su reconocimiento o al menos de una representación lo suficientemente definida y de calidad para que nuestra aplicación sea capaz de detectar que se trata del mismo.

3. Características Exigidas

Interfaz clara y visual

Uno de las características que perseguimos durante el desarrollo del proyecto era que esta presentase una interfaz lo más clara posible con el objeto de que al usuario no le resulte complicado trabajar con ella o le resulte confusa debido, por ejemplo, a la cantidad de elementos presentes el mismo tiempo o la presencia de demasiados menús desplegables. Por ello, hemos limitado la cantidad elementos reconocibles en cada cuadro a 10 para que, debido al tamaño de las pantallas, no le resulte al usuario difícil pulsar sobre el objeto sobre el que esté interesado y no sobre otros próximos al mismo. Con el mismo fin, hemos intentado que estos elementos sean de una tamaño lo mayor posible sin perder la fidelidad con el cuadro, aunque no siempre se ha conseguido mantener en unas dimensiones deseables.

De igual modo, se ha pretendido mostrar los elementos de una forma vistosa y visualmente evidente para que el usuario tenga claro sobre qué aspectos del cuadro puede obtener información y sobre qué zonas no se dispone de ella, aunque al pulsar sobre estas últimas lo que se obtenga sea información sobre el cuadro en general. Se ha optado, pues, por realizar una representación en blanco y negro del cuadro en general con aquellas zonas ocupadas por elementos con información en color y rodeadas por un borde blanco, de esta manera resulta evidente que son estas zonas sobre las que hemos de pulsar para recibir aquella información que corresponda.

De este modo, nos resulta intuitivo el uso de la aplicación y no precisamos una explicación sobre su funcionamiento, sólo con exponer en qué consiste la aplicación nos sería suficiente para que el usuario sea capaz de utilizarla.

Respuesta en tiempo real

Otro de los requisitos indispensables para nuestra aplicación se trata de dar una respuesta en tiempo real a todos los eventos que ocurran, tanto el movimiento de la cámara, como la pulsación sobre un botón en la pantalla de información o sobre la representación del cuadro al ser reconocido, por ejemplo. Este requisito se acentúa de manera evidente al tratarse de Realidad Aumentada, ya que exige que las representación de los elementos 3D y/o otros componentes que han de aparecer como reacción del reconocimiento del cuadro, han de representarse de la manera correcta dependiendo del ángulo, distancia... con el que estamos observando el cuadro.

Así pues, se ha intentado no sobrecargar a la aplicación de los llamados “listeners”, responsables de estar pendiente de eventos sobre la pantalla, para permitir que el tiempo de proceso se dedique a la representación de los objetos 3D, que por lo general suelen ser tareas muy pesadas y costosas.

3. Herramientas

1. Java JDK

Primero, hay que recordar que el lenguaje de programación para el sistema operativo Android se trata de un sublenguaje de Java llamado Dalvik, pese a que hoy en día se le conoce más por su aplicación en Android que por el propio lenguaje, por lo que comúnmente se le llama *Android.* Así pues, el JDK de Java es necesario para el desarrollo de nuestro proyecto.

Fig. 3.1.1 Logotipo de Java

El JDK, Java Developmente Kit, se trata de un conjunto de herramientas software para la creación del programa en lenguaje Java. Existen distribuciones para todos los sistemas operativo mayoritarios.

El JDK contiene los siguientes componentes:

- Intérprete en tiempo de ejecución (java.exe)

  Permite la ejecución de los programas Java (*.class) no gráficos (aplicaciones).

- Compilador (javac.exe)

  Se utiliza para compilar archivos de código fuente Java (habitualmente *.java), en archivos de clases Java ejecutables (*.class). Se crea un archivo de clase para cada clase definida en un archivo fuente.

- Visualizador de applets (appletviewer.exe)

  Es una herramienta que sirve como campo de pruebas de applets, visualizando cómo se mostrarán en un navegador, en lugar de tener que

esperar. Al ser activado desde una línea de órdenes abre una ventana en la que muestra el contenido de la *applet*

- Depurador

Es una utilidad de línea de comandos que permite depurar aplicaciones Java. No es un entorno de características visuales, pero permite encontrar y eliminar los errores de los programas Java con mucha exactitud. Es parecido en su funcionamiento al depurador *gdb*que se incluye con las distribuciones del compilador *gcc/g*++ para C/C++.

- Generador de documentación (javadoc.exe)

Es una herramienta útil para la generación de documentación API directamente desde el código fuente Java. Genera páginas HTML basadas en las declaraciones y comentarios javadoc , con el formato /** comentarios */ :

La documentación que genera es del mismo estilo que la documentación que se obtiene con el JDK.

Las etiquetas, que se indican con una arroba (@), aparecerán resaltadas en la documentación generada.

2. Android ADT

Android Development Tools (ADT) es un plugin para el IDE Eclipse diseñador para proporcionar un entorno potente e integrado en el cual construir aplicaciones Android.

ADT extiende las capacidades de Eclipse para permitirnos una rápida construcción de proyectos Android, crear el entornos o interfaz gráfica, añadir paquetes basados en el Android Framework API, corregir nuestra aplicación utilizando las herramientas del SDK, e incluso exportar nuestras aplicaciones (.apk) para su distribución.

El desarrollo en Eclipse con ADT es muy recomendable y es la manera más rápida para empezar. Con la configuración del proyecto guiada que ofrece, así como la integración de herramientas, los editores de XML personalizados, y el panel de salida de depuración, ADT le da un impulso increíble en el desarrollo de aplicaciones de Android.

3. Android SDK

Se trata del Software Development Kit (SDK) o paquete de desarrollo de software propio de Android para la creación de programas y aplicaciones para el sistema operativo Android. Consta de un depurador de código, bibliotecas necesarias para la programación en el lenguaje, un simulador de teléfonos de diferentes características, documentación, ejemplos de código y tutoriales básicos de programación. A su vez, el SDK nos proporciona un gestor de versiones del propio sistema operativo con el que podemos descargar las librerías adicionales de cada

versión, señalar para qué versión estamos programando y nos notifica si alguno de los métodos usados están obsoletos o todo lo contrario, si en la versión en la que programamos aún no están disponibles. Además, nos permite acceder y controlar dispositivos Android correctamente conectados a nuestro ordenador.

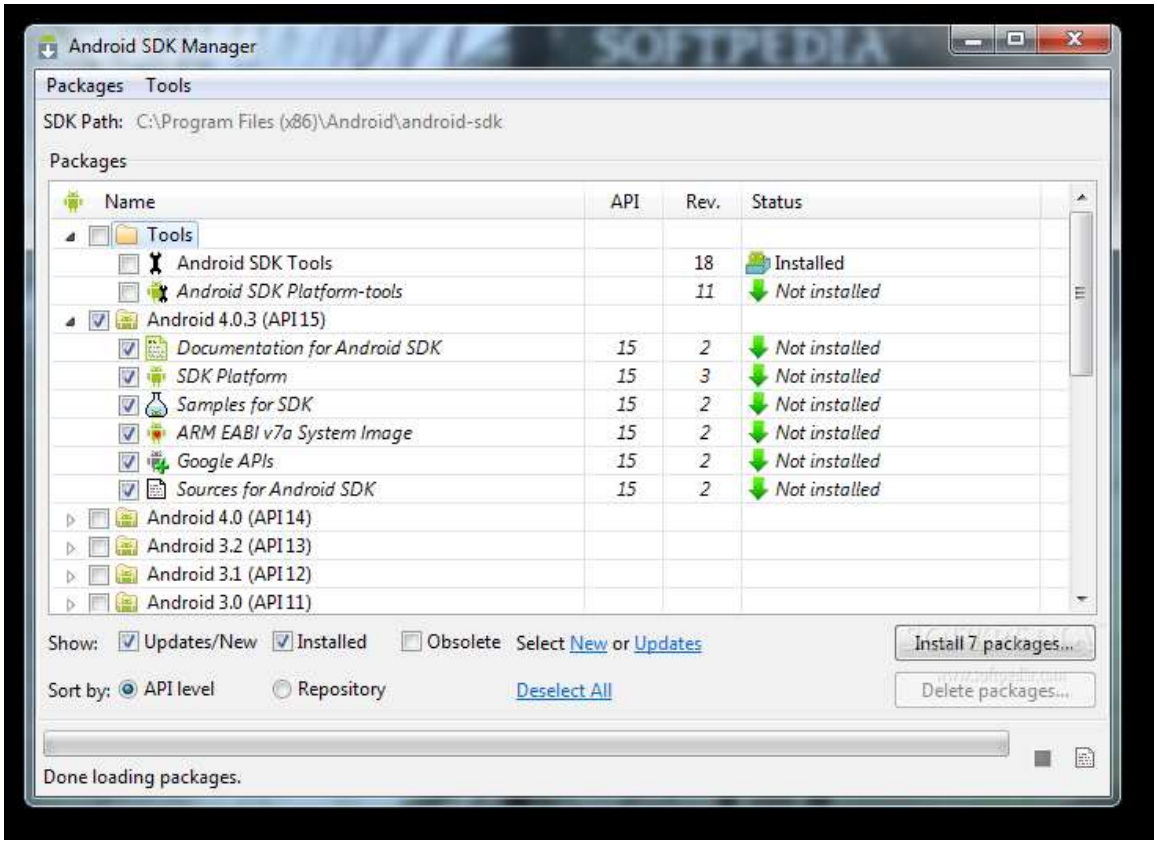


Fig. 3.3.1 Android SDK

4. Android NDK

El NDK (Native Development Kit) de Android es un conjunto de herramientas que permite incorporar los componentes que hacen uso de código nativo en las aplicaciones de Android.

Las aplicaciones para Android se ejecutan en la máquina virtual Dalvik. El NDK permite implementar parte de tus aplicaciones usando código nativo con lenguajes como C y C + +. Esto puede proporcionar beneficios a ciertas clases de aplicaciones, en la medida que se puede reutilizar el código existente y en algunos casos obtener un aumento de la velocidad.

El NDK establece lo siguiente:

- Un conjunto de herramientas y archivos de construcción que se utilizan para generar bibliotecas de código nativo de fuentes en C y C + +.
- Una manera de integrar las correspondientes bibliotecas nativas en un archivo de paquete de aplicaciones (.apk) que se pueden implementar en los dispositivos Android.

- Un conjunto de cabeceras y bibliotecas nativas del sistema que estarán soportadas en todas las futuras versiones de la plataforma Android, a partir de Android 1.5. Las aplicaciones que utilizan actividades nativas deben de ejecutarse sobre Android 2.3 o posterior.
- Documentación, ejemplos y tutoriales.

5. Notepad++

Notepad++ es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación exclusivo del sistema operativo Windows. Presenta similitudes con el bloc de notas estándar de Windows pero con ciertas características añadidas que lo hacen más fácil de usar y cómodo.

- Coloreado y envoltura de sintaxis: si se escribe en un lenguaje de programación o marcado, Notepad++ es capaz de resaltar las expresiones propias de la sintaxis de ese lenguaje para facilitar su lectura.
- Pestañas: al igual que en muchos navegadores, se pueden abrir varios documentos y organizarlos en pestañas.
- Resaltado de paréntesis e indentación: cuando el usuario coloca el cursor en un paréntesis, Notepad++ resalta éste y el paréntesis correspondiente de cierre o apertura. También funciona con corchetes y llaves.
- Grabación y reproducción de macros.
- Soporte de extensiones: incluye algunas por defecto.

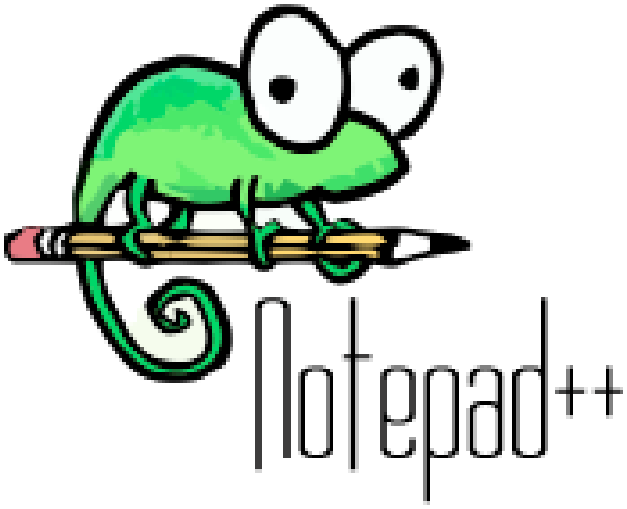


Fig.3.5.1 Logotipo de Notepad++

Da soporte a una gran cantidad de lenguajes de programación, además de permitir al usuario crear reglas de coloreado y demás características para el suyo propio. Nosotros lo hemos utilizado para la edición del código C++.

6. Eclipse

Según su propia página se trata de “una comunidad de particulares y organizaciones que quieren colaborar en un software abierto, comercial y amigable. Este proyecto está centrado en la construcción de una plataforma de desarrollo compuesta por frameworks extensibles, herramientas y runtimes para la construcción, despliegue y manejo de software durante su ciclo de vida.”

En la práctica, Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que llaman "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java DevelopmentToolkit(JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Posteriormente fue liberado originalmente bajo la Common Public License, para después ser re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU (GNU GPL).

Fig.3.6.1 Logo Eclipse

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Mediante diversos plugins estas herramientas están también disponibles para otros lenguajes.

7. Cygwin

Cygwin se define de manera muy esquemática y escueta en la página principal del programa y además nos aclara que no es.

Cygwin es:

- Una colección de herramientas que proporcionan una apariencia y entorno de Linux dentro de Windows.
- Un DLL (cygwin1.dll) que actúa como una capa del API de Linux proporcionando sustancialmente las funcionalidad del mismo API.

Cygwin no es:

- Una manera de correr o ejecutar aplicaciones nativas de Linux en Windows. Debes reconstruir el código de tu aplicación si quieres que funcione en Windows.
- Una manera mágica de hacer que tus aplicaciones nativas de Windows este preparadas para las funcionalidades de UNIX como señales, ptys, etc. De nuevo, necesitas reconstruir tus aplicaciones si quieres aprovecharte de las ventajas de las funcionalidad de Cygwin

Fig.3.7.1 Logo de Cygwin

En la práctica esto viene a ser que podremos usar una consola de comandos como la de linux, y utilidades como “ftp”,”telnet”,”ls”,”ps”, otras “tools” para comprimir y descomprimir archivos, editores como “vi” y “joe”, apache, ssh, “gcc” para poder compilar, e incluso nuestro propio servidor X. Lo que lo hace “ideal” si tienes que trabajar con otras máquinas Unix/Linux desde tu estación de trabajo o incluso para escribir y probar shell scripts, o ir aprendiendo los comandos básicos de Linux.

8. Qualcomm y Vuforia

Qualcomm, según su propia página, es “El líder mundial en la próxima generación de tecnologías móviles,las ideas e innovaciones de Qualcomm han impulsando el crecimiento inalámbrico y ayudar a conectar a las personas a la información, el entretenimiento y los otros.”

Pero sus campos de investigación y trabajo van mucho más allá de la tecnología inalámbrica. Es una compañía de semiconductores que diseña, manufactura y comercializa tecnología de hardware y software desde su base de San Diego, California. Entre todo ello, cabe destacar el hecho de ser el mayor fabricante de chips y procesadores para smartphones del mundo. Así como el hecho de ser la compañía responsable del desarrollo y difusión de los estándares de transmisión de datos en dispositivos móviles CDMA y W-CDMA entre otros. Además son los dueños del cliente de correo electrónico Eudora. Es pues, una de las mayores compañías del sector informático tanto en el desarrollo de tecnología software como en hardware.

Fig. 3.8.1 Logo de Qualcomm

Así, nos encontramos que hace unos años Qualcomm decidió crear su propio sistema de realidad aumentada con el que proporcionar las herramientas (API y SDK) necesarias a los desarrolladores para que puedan construir de forma libre todas las

aplicaciones y programas que deseen, de hecho tuvo un crecimiento exponencial y fue incorporada a más de 1000 aplicaciones durante el último año. Actualmente, Vuforia cuenta con más de 30 mil desarrolladores registrados de más de 130 países y esto ha permitido impulsar una nueva generación de experiencias móviles que encantan a todos los usuarios y otorgan un valor agregado a la publicidad y a los medios impresos, los productos o envases de consumo y materiales con fines educativos.

En sus últimas actualizaciones permitirá el uso de la tecnología de reconocimiento de imagen en la nube. Hasta hoy, las aplicaciones de Vuforia realizaban el reconocimiento de imágenes mediante el uso de bases de datos locales de hasta 80 imágenes. El uso de la potencia de la nube posibilita que el reconocimiento se realice con bases de datos de más de un millón de imágenes. Esta nueva función permitirá nuevas experiencias comerciales móviles que beneficiarán a los consumidores, desarrolladores, minoristas y promotores de marca.

9. Target Management System

Se trata de la herramienta online proporcionada por la arquitectura Vuforia para la creación de los patrones de reconocimientos utilizados por el SDK Vuforia para la identificación de los objetos y su posterior tratamiento. Para ello, la realidad aumentada basada en el visionado, a través de cualquier dispositivo con cámara, construida con este SDK, debe tener lo que se conoce como *"DataSet"* con el que comparar la imagen captada por la cámara . El *"Target Management System"* ofrece una herramienta basada en la web para los desarrolladores que utilizan el SDK Vuforia para crear esos *"DataSet"* a partir de una imagen importada a la herramienta. El objeto que nos crea, se quedará almacenado en la página de Vuforia y podremos descargarnos para su utilización en nuestra aplicación.

Para acceder a la herramienta por lo tanto, primero habremos de registrarnos en la página de Vuforia como desarrolladores del mismo.

10. Instalación de las Herramientas y Preparar el Dispositivo

La instalación de las herramientas necesarias para el desarrollo del proyecto se deben realizar en el orden expuesto a continuación para su correcto funcionamiento, si alteramos el mismo puede darnos problemas y no funcionar de forma correcta o simplemente resultar imposible al no disponer de ficheros y/o directorios que deberían existir llegados a ese punto.

Para el desarrollo de nuestro proyecto hemos de usar Windows XP, Vista o 7, pese a que muchas de las herramientas se pueden instalar en otros SO.

1. Notepad ++

Este programa no presenta ninguna peculiaridad especial respecto a cualquier otro programa de Windows. Igualmente estos son los pasos:

1) Descargamos la última versión de:

   http://notepad-plus-plus.org/download/v6.1.8.html

2) Ejecutamos el fichero descargable y seleccionamos el idioma.

Fig.3.10.1.1 Lenguaje de Instalación

3) Seleccionamos el directorio de instalación.

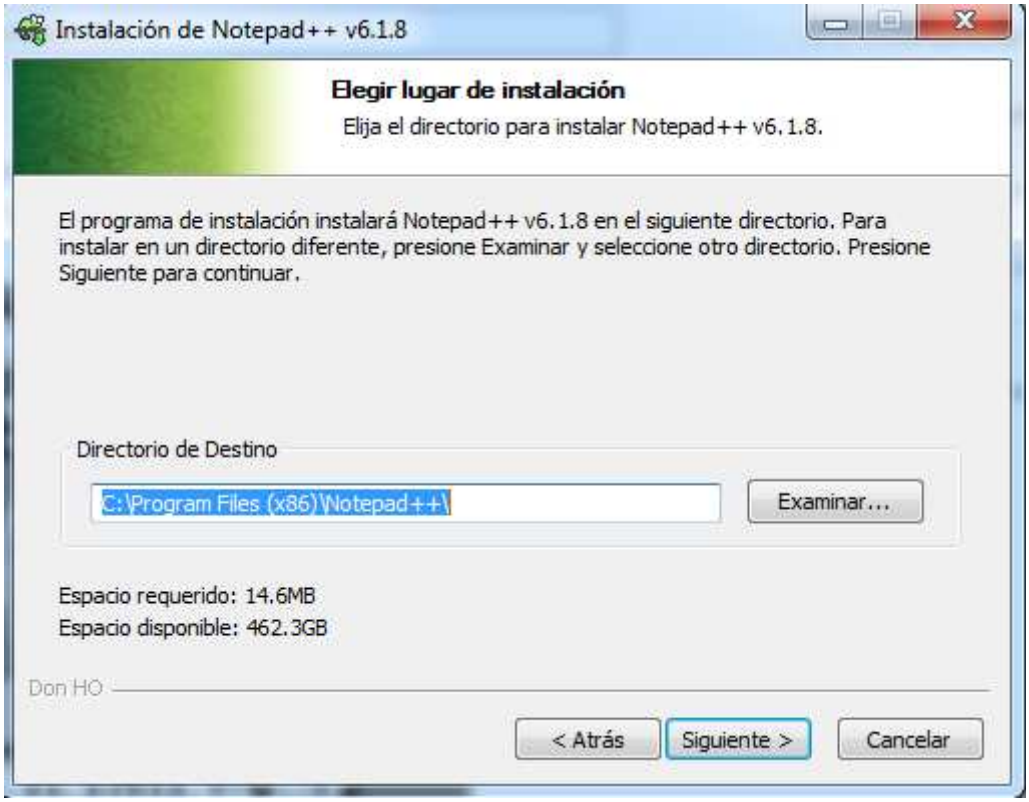


Fig.3.10.1.2 Ruta de Instalación

4) Le damos a siguiente hasta que nos aparece la opción de debajo, donde pulsamos crear un icono en el Escritorio.

Fig.3.10.1.3 Crear icono en el escritorio.

5) Le damos a siguiente hasta que termina su instalación.

2. Instalación JDK

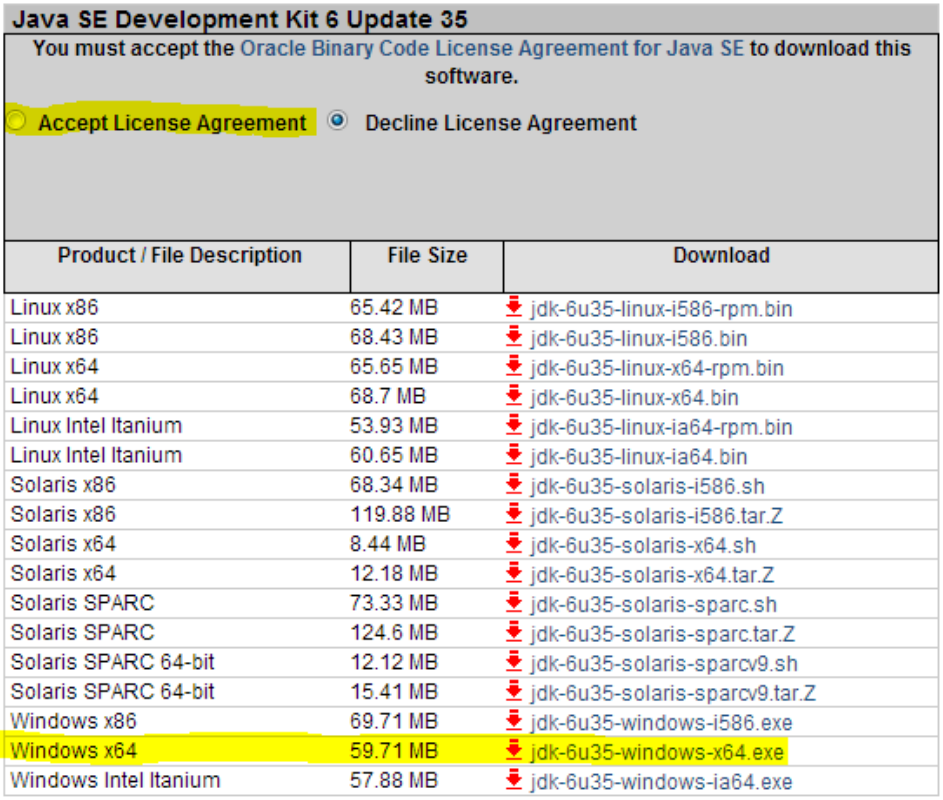
El primer paso para conseguir construir nuestro proyecto es bajarnos el JDK de la página oficial. Debido a que la versión 7 salió al público con posterioridad al comienzo del proyecto, nosotros trabajamos con la anterior, que además estará más testeada y con menos errores.

1) Accedemos al enlace de abajo siguiente:

http://www.oracle.com/technetwork/java/javase/downloads/jdk6u35-downloads-1836443.html

2) Aceptamos la licencia.

3) Seleccionamos el sistema operativo que tenemos en el equipo en el que vamos a trabajar, en nuestro caso Windows 64bits.



Java SE Development Kit 6 Update 35

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux x86	65.42 MB	jdk-6u35-linux-i586-rpm.bin
Linux x86	68.43 MB	jdk-6u35-linux-i586.bin
Linux x64	65.65 MB	jdk-6u35-linux-x64-rpm.bin
Linux x64	68.7 MB	jdk-6u35-linux-x64.bin
Linux Intel Itanium	53.93 MB	jdk-6u35-linux-ia64-rpm.bin
Linux Intel Itanium	60.65 MB	jdk-6u35-linux-ia64.bin
Solaris x86	68.34 MB	jdk-6u35-solaris-i586.sh
Solaris x86	119.88 MB	jdk-6u35-solaris-i586.tar.Z
Solaris x64	8.44 MB	jdk-6u35-solaris-x64.sh
Solaris x64	12.18 MB	jdk-6u35-solaris-x64.tar.Z
Solaris SPARC	73.33 MB	jdk-6u35-solaris-sparc.sh
Solaris SPARC	124.6 MB	jdk-6u35-solaris-sparc.tar.Z
Solaris SPARC 64-bit	12.12 MB	jdk-6u35-solaris-sparcv9.sh
Solaris SPARC 64-bit	15.41 MB	jdk-6u35-solaris-sparcv9.tar.Z
Windows x86	69.71 MB	jdk-6u35-windows-i586.exe
Windows x64	59.71 MB	jdk-6u35-windows-x64.exe
Windows Intel Itanium	57.88 MB	jdk-6u35-windows-ia64.exe

Fig.3.10.2.1 Elección del Sistema y Aceptación de la Licencia.

4) Una vez descargado el instalador, lo ejecutamos y dejamos que se instale todo con los valores por defecto.

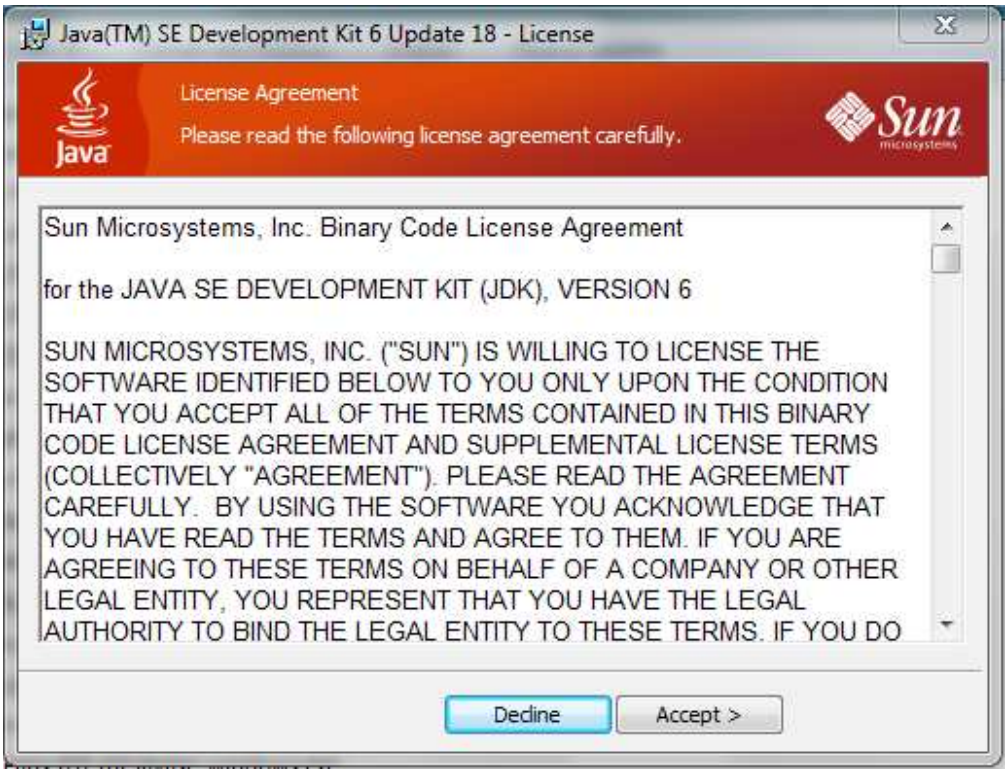


Fig.3.10.2.2 Aceptación de las Condiciones

5) Ya tenemos el JDK instalado en nuestro equipo.

3. Instalación de Eclipse

Una vez tenemos el JDK instalado, procedemos a bajar e instalar el IDE Eclipse, para ello seguimos pasos similares:

1) Accedemos a la página de descarga:

http://www.eclipse.org/downloads/

2) Escogemos **Eclipse IDE for Java EE Developers** y pinchamos en la versión de 64bits en nuestro vaso.

Fig.3.10.3.1 Paquete y Sistema Operativo

3) Una vez descargado, descomprimimos el contenido en la carpeta donde queramos que quede el programa.

4) Ejecutamos el fichero *eclipse.exe*

5) Al ejecutarlo por primera vez o si no marcamos la opción marcada, nos pedirá que asignemos el espacio de trabajo (workspace) donde se sitúan todos nuestro proyectos futuros. Sin embargo, veremos cómo podemos trabajar con proyecto fuera de este espacio de trabajo.

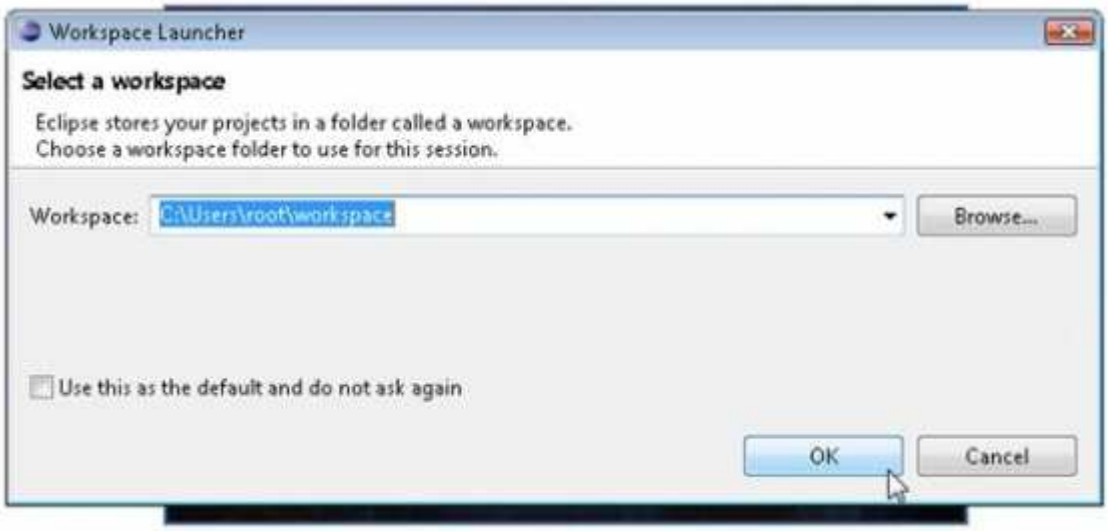


Fig.3.10.3.2 Ruta de Instalación

6) Ya tenemos el entorno de desarrollo funcional.

4. Instalación del Android SDK Downloader

Ahora, disponemos ya de las herramientas necesarias para programar en Java y un entorno de desarrollo para hacerlo de forma cómoda. Nos toca pues, descargar el SDK de Android que nos añada las herramientas para trabajar con Android.

1) El SDK de Android se distribuye dentro de un paquete de herramientas de comienzo en:

   http://developer.android.com/sdk/index.html

2) Desempaquetados y copiamos el contenido en un directorio, por ejemplo en: *"C:\Development\Android\android-sdk-windows\".* Importante que no contenga espacios la ruta. A partir de ahora trabajamos con la siguiente igualdad:

   <DEVELOPMENT_ROOT>= C:\Development\Android

3) Ahora hemos de añadir la ruta de tools\ al path de Windows, para ello seguimos los siguientes pasos:
   a) Vamos al menú inicio de Windows y sobre *"Equipo"* hacemos click con el botón derecho del ratón.
   b) Propiedades

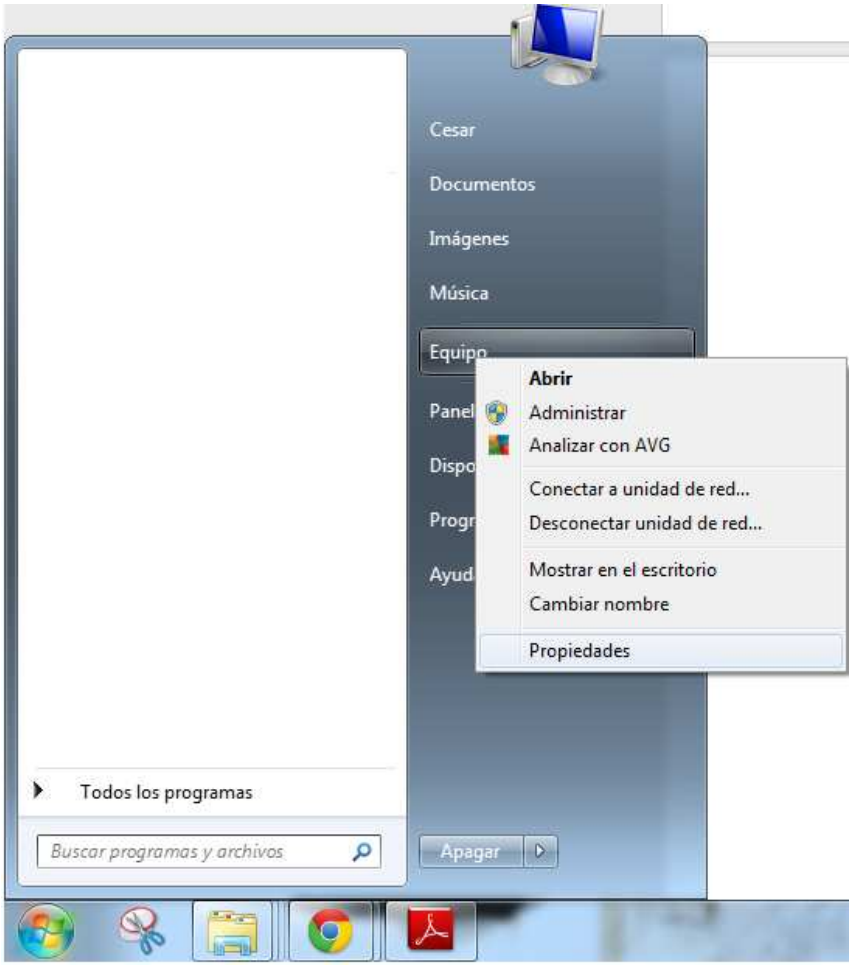


Fig.3.10.4.1 Propiedades del Equipo

c) Pulsamos en el margen izquierdo la opción *"Configuración Avanzada del Sistema"*

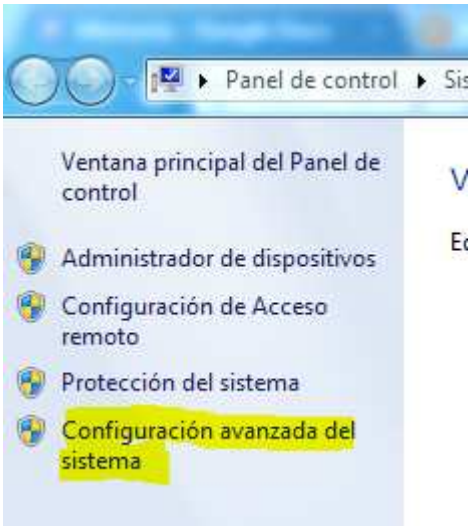


Fig.3.10.4.2 Configuración avanzada

d) En la pestaña “Opciones Avanzadas” pulsamos sobre “Variables del Entorno”

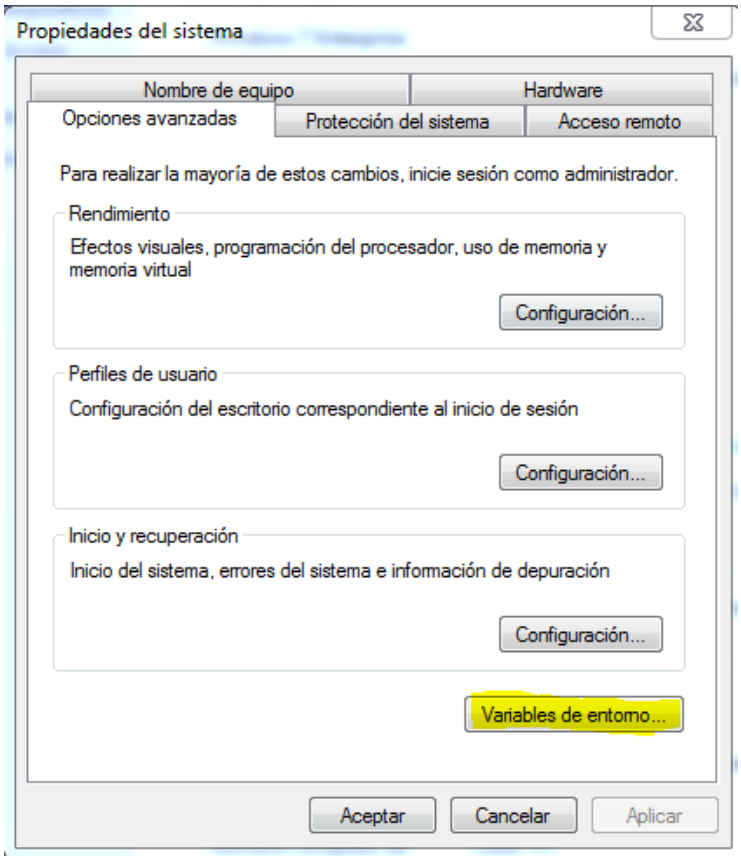


Fig.3.10.4.3 Variables de entorno

e) En el recuadro inferior, buscamos las variables “*Path”* y le damos a editar.

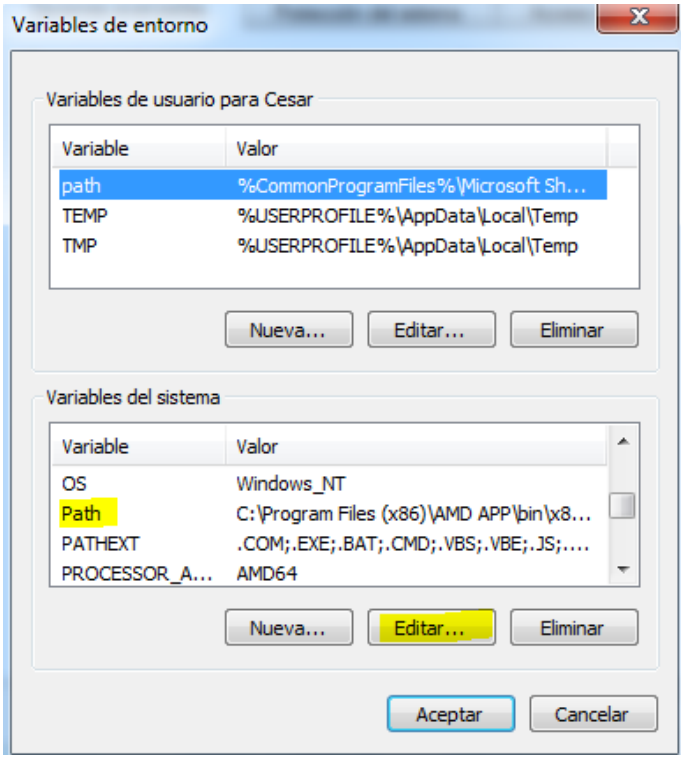


Fig.3.10.4.4 Editar Path

f) Por último añadimos nuestra ruta donde hemos descomprimido el SDK al final de la línea utilizando punto y coma como separador:

C:\Development\Android\android-sdk-windows\tools\

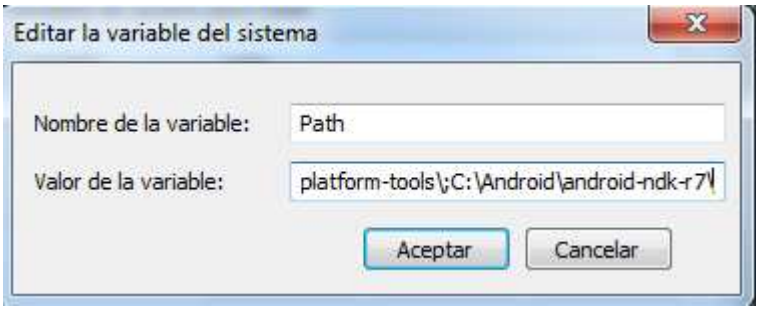


Fig.3.10.4.5 Añadimos la ruta.

4) Ya tenemos funciona el SDK dentro de nuestro sistema.

La instalación de Android ADT en el siguiente paso utilizará esa ruta para identificar el entorno de desarrollo de Android.

5. Android ADT

Lo que vamos hacer ahora es conectar Eclipse con el SDK de Android. Para ello procedemos de la siguiente forma:

1) Abrimos Eclipse
2) Seleccionamos Help ->Install New Software

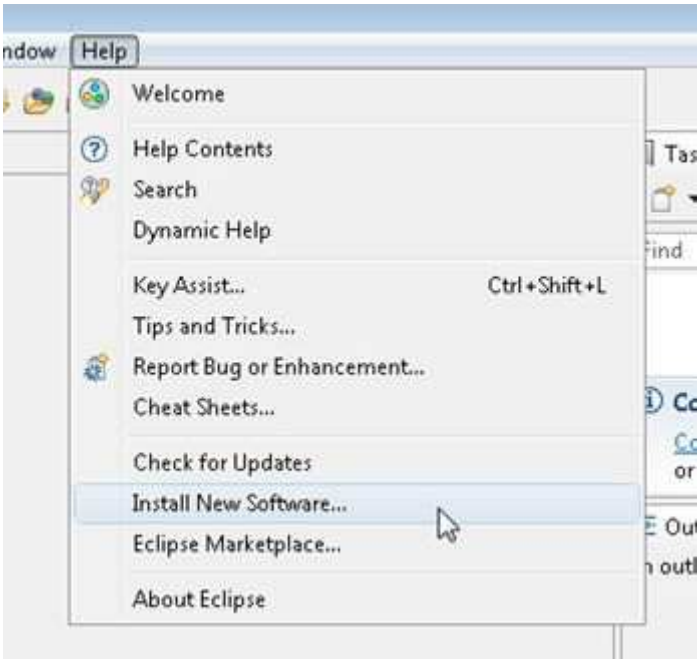


Fig.3.10.5.1 Install New Software

3) Pulsamos el botón “Add” situado en la esquina superior derecha e introducimos los siguientes parámetros en la nueva ventana emergente.

Name: ADT Plugin
Location: https://dl-ssl.google.com/android/eclipse/

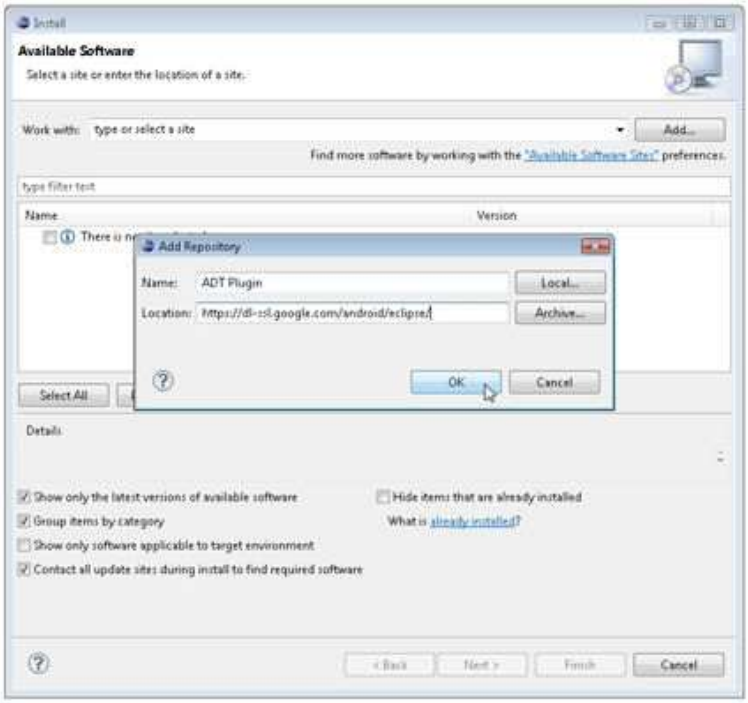


Fig.3.10.5.2 Añadir nuevo repositorio

4) Al pulsar sobre OK nos aparecerán un listado en medio de la ventana con los componentes disponibles, los seleccionamos y pulsamos sobre “*Next”.*

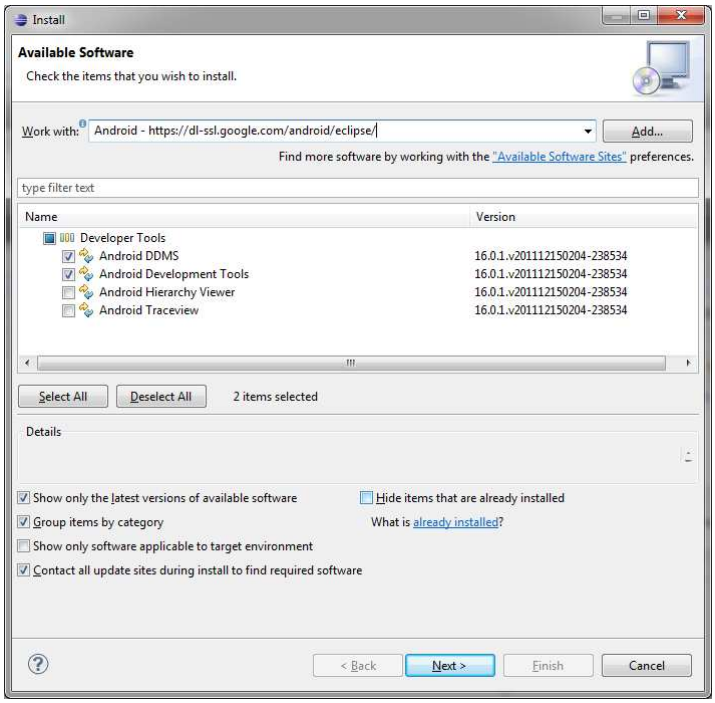


Fig. 3.10.5.3 Selección de herramientas

5) Aceptamos todas las licencias y acabamos de instalar el ADT

6. Android SDK Platform Support

Para el desarrollo de aplicaciones Android en la correcta versión debe estar instalada la plataforma correcta. Para ello usamos la "Android SDK Manager". Con ella instalamos componentes y soportes adicionales para cada versión.

1. Seleccionamos en Eclipse Window -> Android SDK Manager.
    a. En caso de que el SDK no esté instalado en la ruta correcta hay que ir a Windows->Preferences->Android y ponemos el campo del SDK a la ruta correcta.
2. En la ventana del SDK Manager hacemos click sobre Deselect All y seleccionamos a continuación los siguientes paquetes, la imagen puede no corresponder ya que las sucesivas actualizaciones de Android han modificado la jerarquía pero aun así nos sirven como guía para la instalación:
    - De Tools:
        - Android SDK Platform-tools
    - De Android 4.0 (API 14):
        - Documentation for Android SDK
    - De Android 2.3.3 (API 10):
        - SDK Platform
    - Samples for SDK (opcional)
    - De Extras:
        - Google USB Driver package

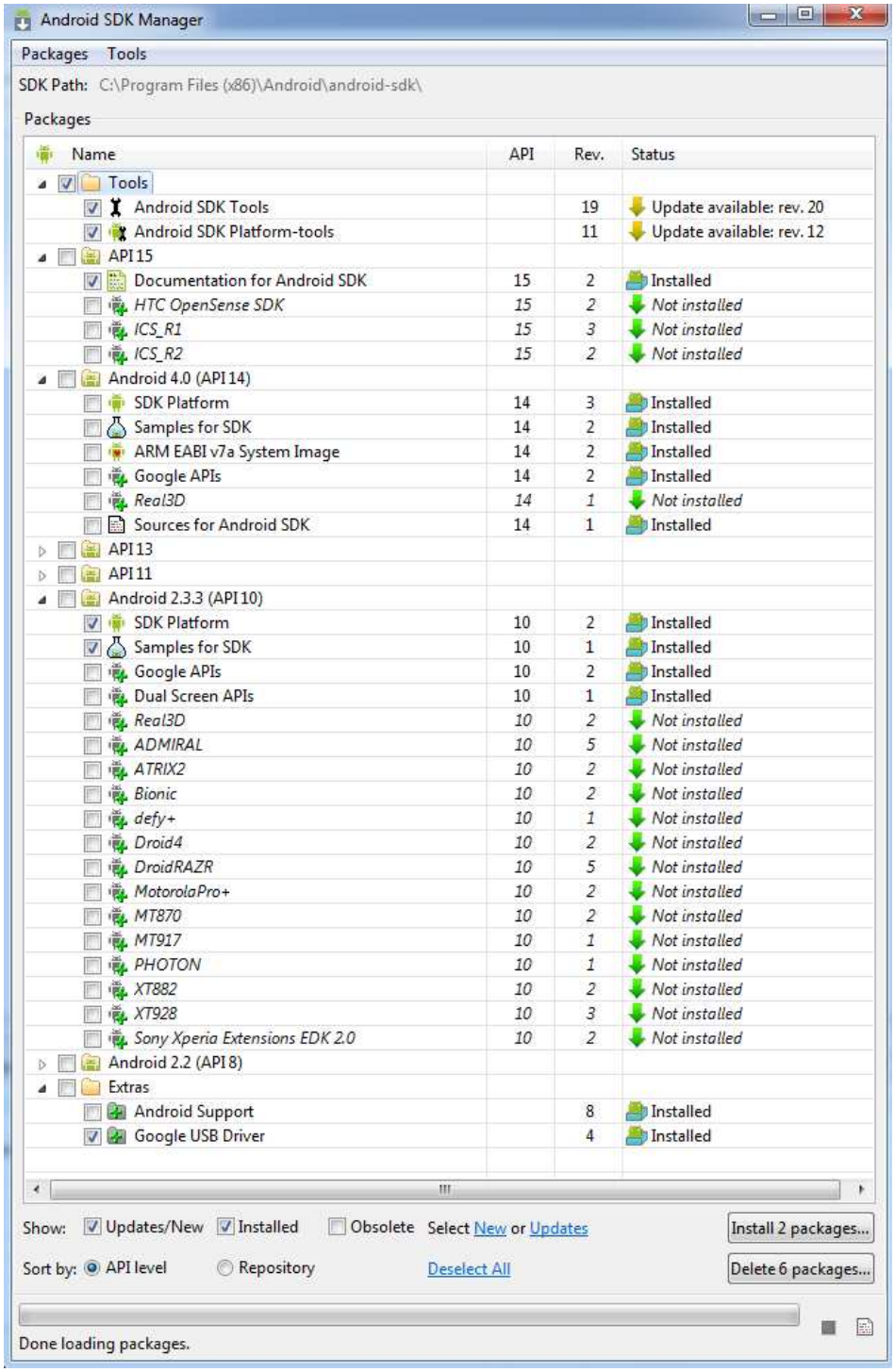


Fig.3.10.6.1 Selección de paquetes.

3. Para instalar las opciones seleccionadas hay darle a “*AcceptAll”* y luego a *“Install”.*

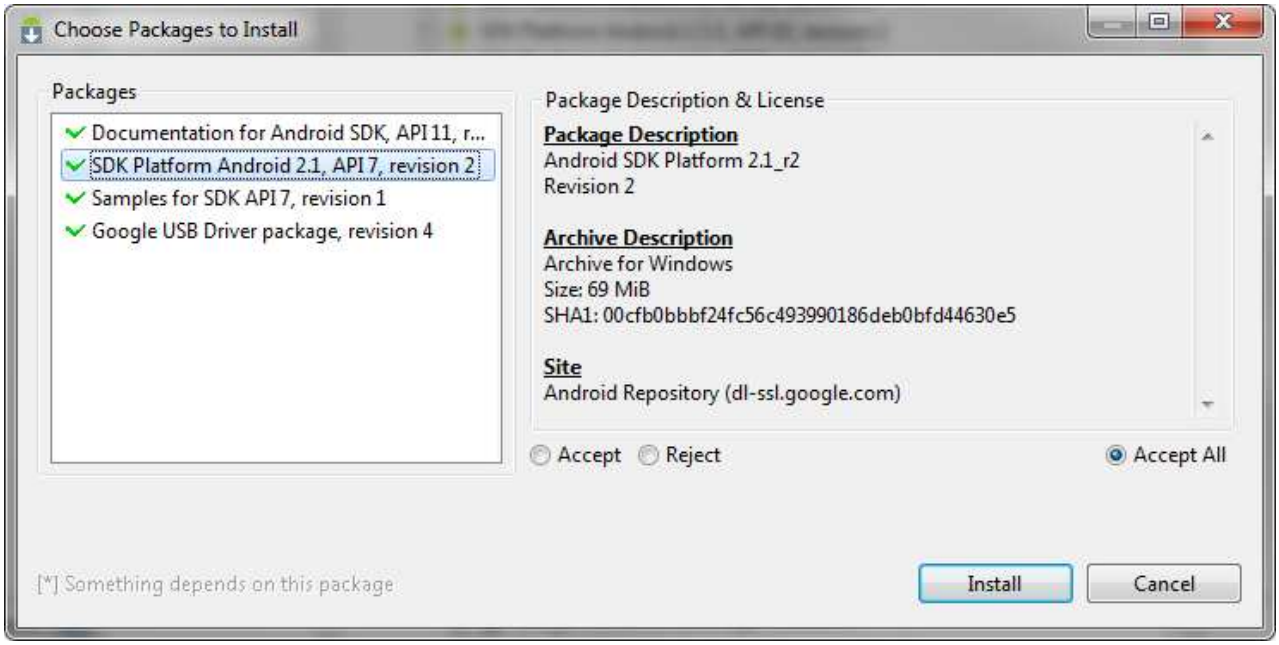


Fig.3.10.7.1 Selección de paquetes.

4. Por último, añadimos, siguiendo los pasos antes explicados, la siguiente ruta al path de Windows:

C:\Development\Android\android-sdk-windows\platform-tools

7. Instalación de Cygwin

Se requiere un compilador GNU tanto las aplicaciones dinámicas como las librerías para el NDK de Android. El comando *make* está diseñado para funcionar con *gcc4.* En Windows, un forma conveniente de tener el entorno preparado es instalar Cygwin.

Cygwin utiliza un asistente de instalación para controlar el proceso de instalación. Hay que seguir los siguientes pasos.

1) Ir a http://www.cygwin.com/setup.exe
2) Seleccionar “Install from the Internet”

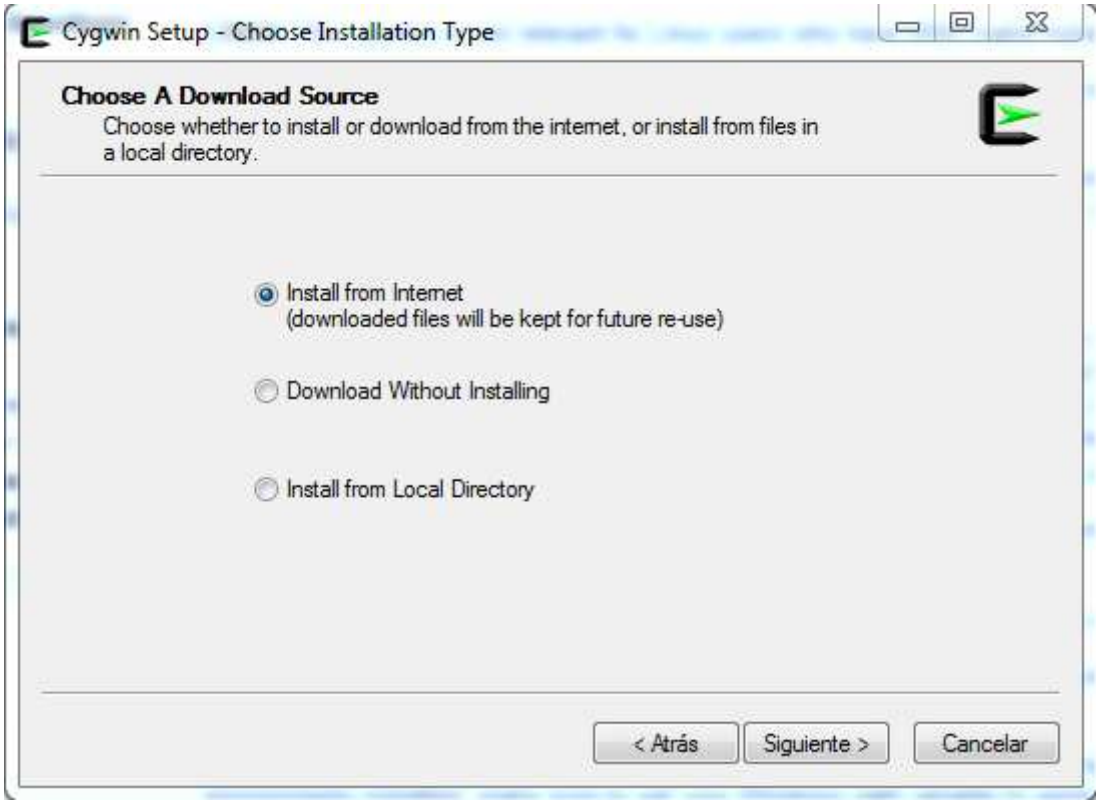


Fig.3.10.7.1 Instalar desde internet

3) En la siguiente etapa se recomienda no cambiar el directorio raíz y dejarlo en “C:\cygwin”
4) En "Local Package Directory" se almacenarán los paquetes descargados, así que igual quiere mantenerlo en la misma carpeta de instalación del programa.
5) Seleccionamos un sitio de descarga con descarga rápida. Como no conocemos ninguno en especial, seleccionamos uno cualquiera.

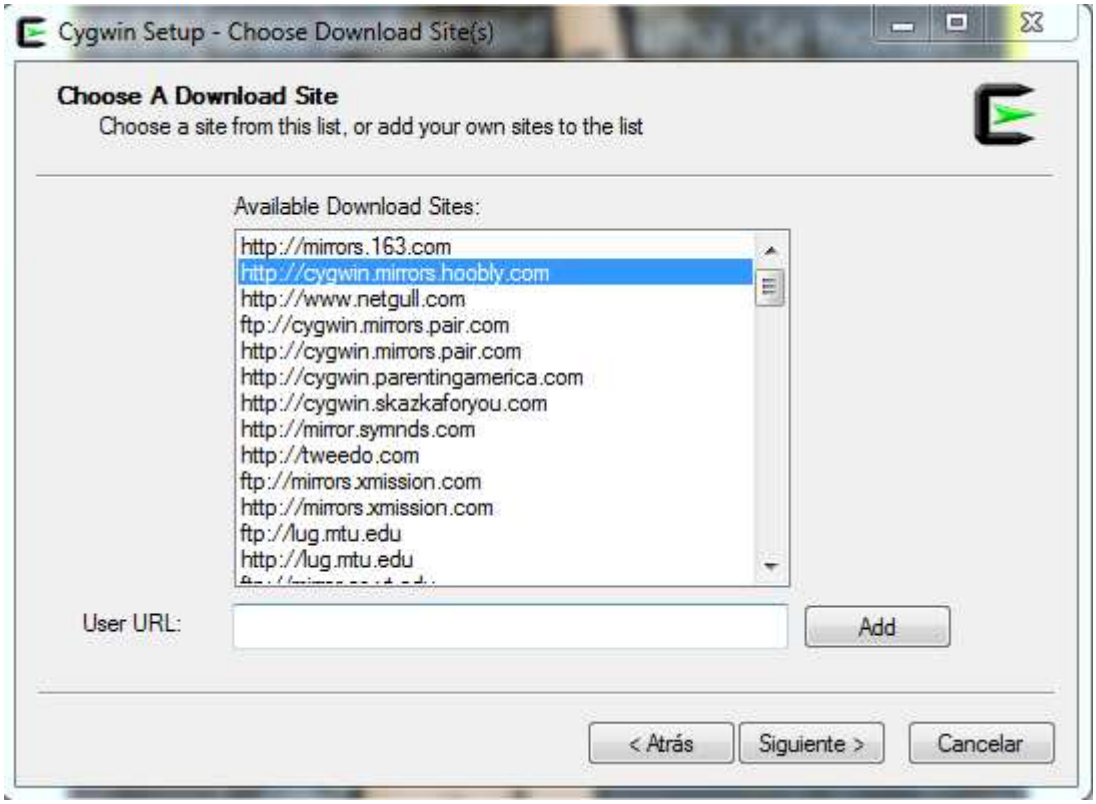


Fig. 3.10.7.2 Selección de sitio de descarga.

6) Cuando se descarguen los paquetes de información, veremos una jerarquía para elegir diferentes paquetes a descargar.
   a) Seleccionamos All ->Devel -> "make: The GNU version of the 'make' utility".
   b) Pinchamos donde pone *"skip"* hasta que salga la versión.
7) Finalizamos la instalación haciendo clic en *"Next"*
8) Ya tenemos el entorno Cygwin totalmente instalado

8. Instalación del Android NDK

Para instalar esta extensión del SDK de Android con el fin de trabajar con código nativo hay que seguir los pasos:

1) Descargar el NDK de la página

   http://developer.android.com/sdk/ndk/index.html

2) Desempaquetar el contenido en un directorio. Siguiendo con lo dicho hasta ahora, lo haremos en el directorio *"C:\Development\Android\android-ndk-r7\".* Para que el SDK y el NDK de Android comparten el mismo directorio padre.
3) Añadimos, como antes hicimos con otras rutas, ese directorio al path de Windows
4) Ya tenemos el NDK instalado y listo para usarse.
5) Para comprobar su correcta instalación hacemos lo siguiente:
   a) Usando la consola de Cygwin, navegamos hasta el directorio de instalación.
   b) Vamos a la carpeta "samples"

c) Entramos en cualquiera de los ejemplos. En nuestro caso “san-angeles”
d) Ejecutamos el comando “ndk-build”
e) El ordenador debería haber producido una librería dinámica “libssanangeles.so” y haberla escrito en “/libs/armeabi”.

9. Instalación del Vuforia SDK

Ahora vamos a instalar el SDK de Vuforia, que es el que trabaja con el API sobre el que hemos trabajado y del que hemos sacado el acceso a la cámara y el reconocimiento de la imagen. Para ello, nuevamente, hemos de seguir los pasos listados:

1) Descargamos el SDK distribuido en la siguiente página:

   https://ar.qualcomm.at/qdevnet/
2) Ejecutamos el fichero descargado.

Fig.3.10.9.1 Ejecución del fichero

3) Pinchamos sobre Next.

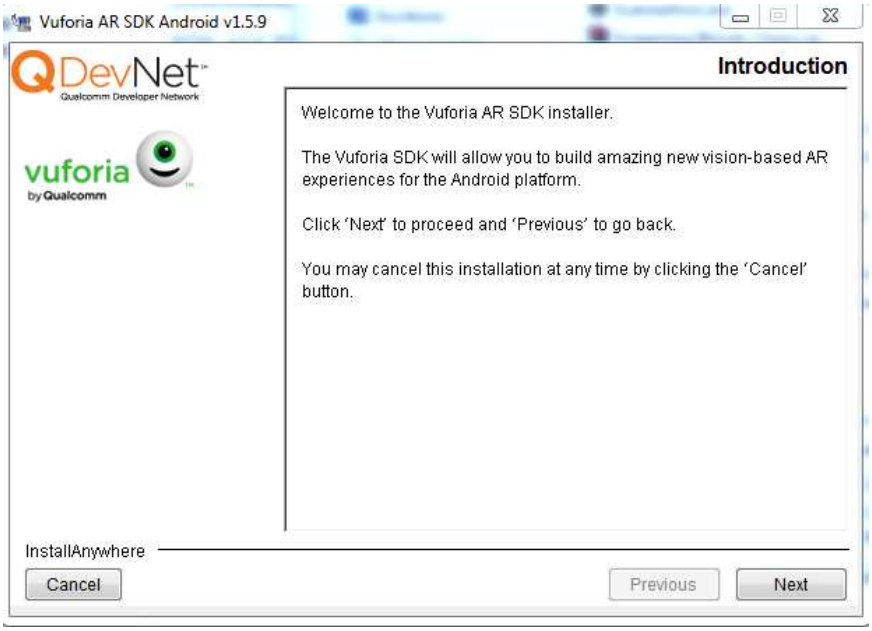


Fig.3.10.9.2 Pantalla de inicio del instalador

4) Introducimos la ruta donde queremos instalarlo. En nuestro caso lo hacemos en una carpeta con el mismo padre que todo lo de Android.

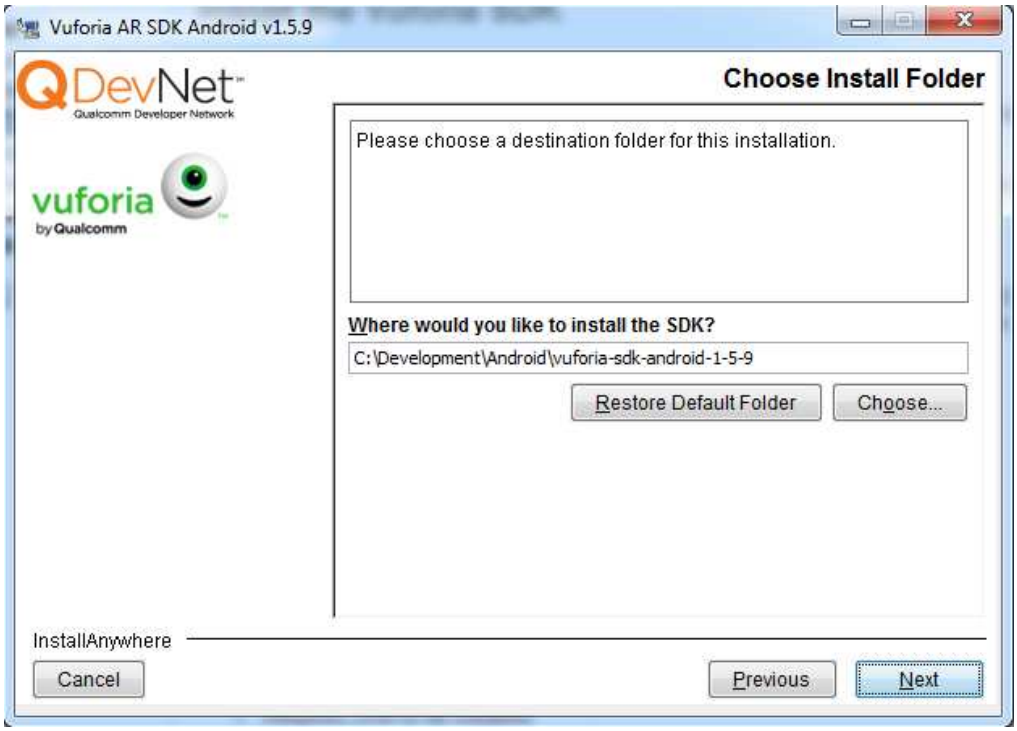


Fig.3.10.9.3 Selección de paquetes.

5) Bajamos al final del texto de la licencia, ya que de otra forma no nos deja aceptar la licencia, y la aceptamos.

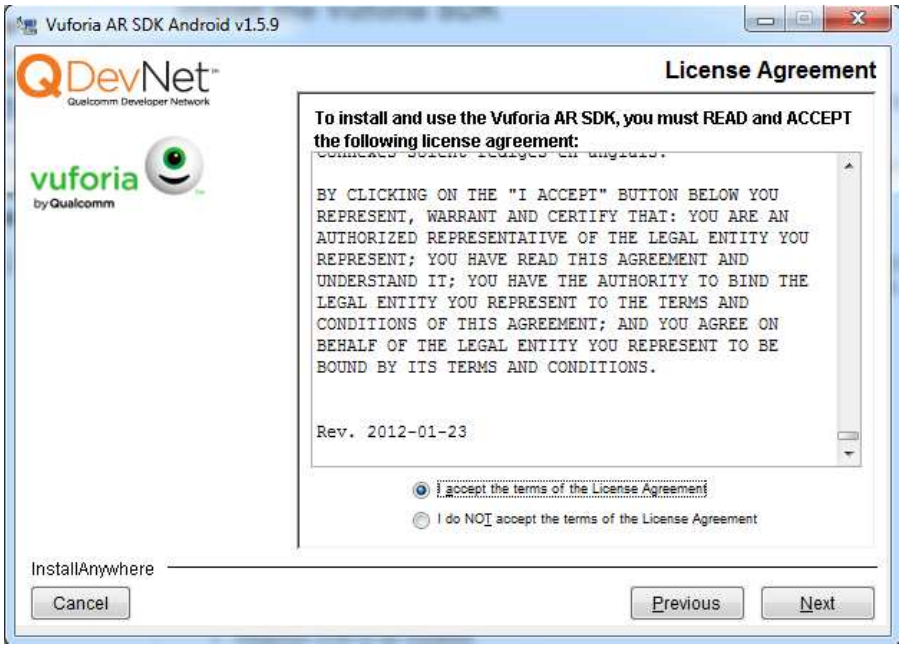


Fig.3.10.9.4 Aceptación del acuerdo de licencia.

6) Presionamos sobre *Install* y se instalarán los paquetes en el directorio indicado.

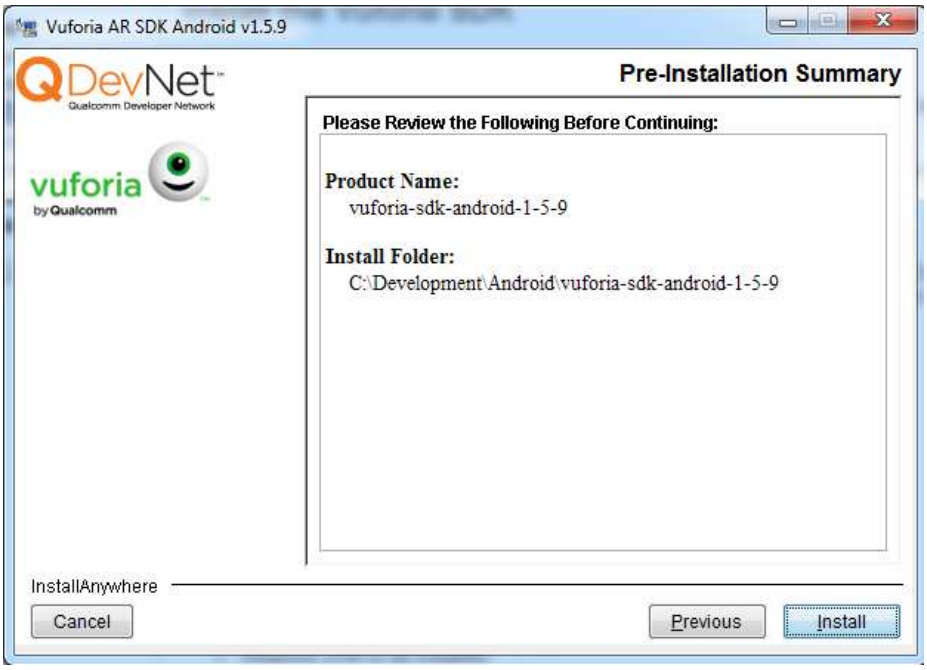


Fig.3.10.9.5 Selección de paquetes.

7) Añadimos el directorio al entorno de trabajo de Eclipse. Para ello:
   a) Entramos en Eclipse.
   b) Vamos a Window->Preferences
   c) Navegamos en el menú hasta Java->BuildPath->Classpath Variables

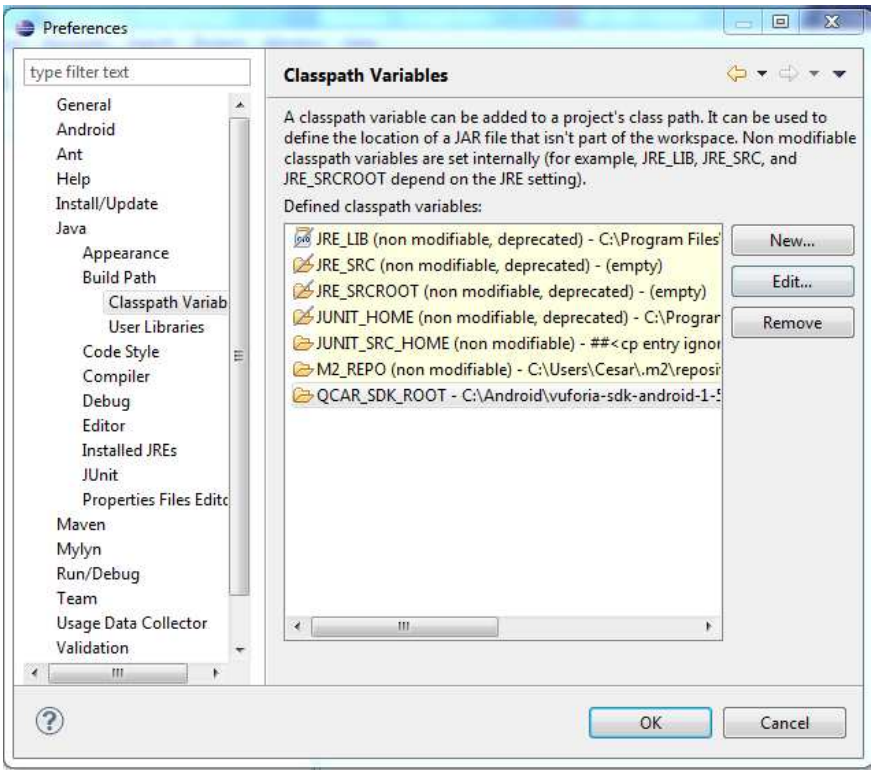


Fig.3.10.9.6 Añadir Classpath.

d) Creamos una nueva variable con los datos, donde xx-yy-zz es la versión del SDK:

Name: QCAR_SDK_ROOT
Folder:<DEVELOPMENT_ROOT>\vuforia-sdk-android-xx-yy-zz

e) Le damos a OK

8) Ya tenemos asignada la variable al entorno de trabajo Eclipse.
9) Por último, hemos de dar un paso más antes de poder ejecutar, para empezar, los ejemplos contenidos en el SDK. Preparar el dispositivo, para ello:
   a) En nuestro dispositivo Android, vamos a Settings->Applications (Ajustes->Aplicaciones)
   b) Marcamos "Unknownsources" y aceptamos el mensaje de advertencia que nos sale.

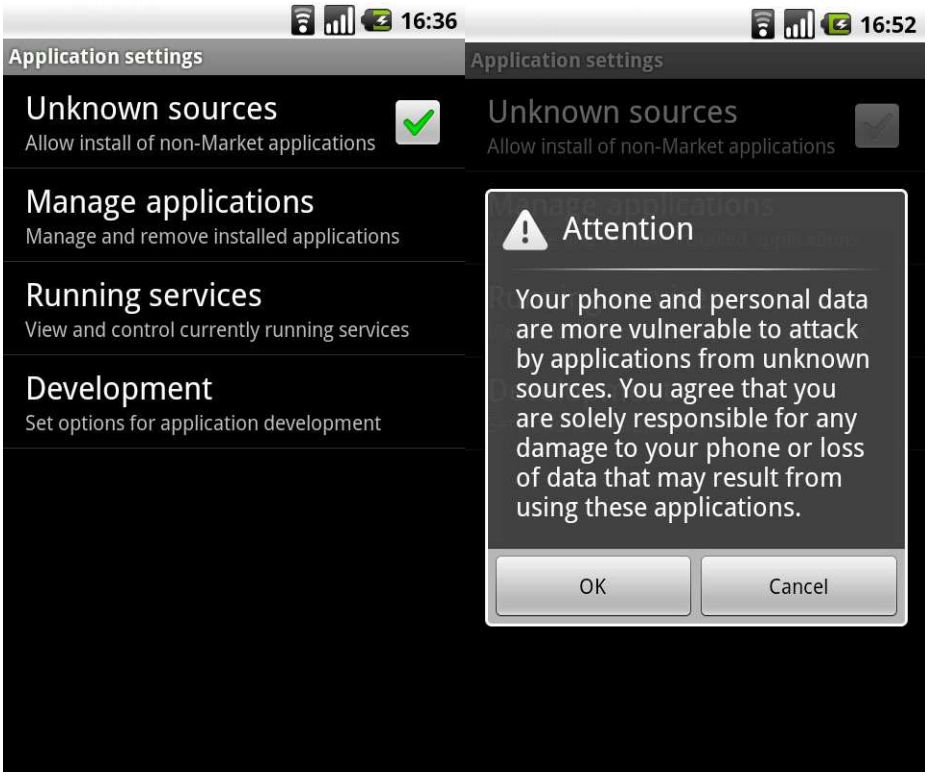


Fig.3.9.9.7 Habilitar instalación de fuentes desconocidas (distintas del market)

c) Vamos a la sección Development (Desarrollo) del mismo menú.

d) Marcamos las opciones de la imagen.

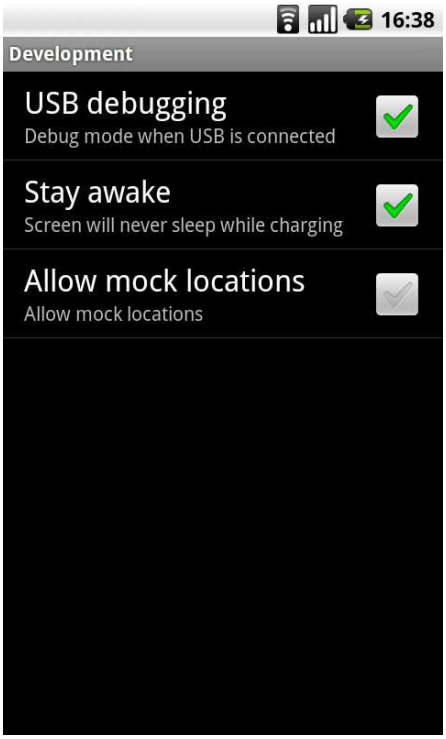


Fig.3.9.9.8 Habilitar depuración de USB y permaneces desbloqueado al estar conectado.

e) Ya tenemos el dispositivo listo para recibir aplicaciones desde el ordenador.

Target ManagementSystem

Se trata del sistema de Vuforia para la creación de los conocidos como *DataSet* que no son ni más ni menos que un conjunto de patrones o puntos detectados por el sistema con los cuales es capaz de realizar el reconocimiento de nuestros objetos.

Para ello hemos de seguir unos pasos muy sencillos:

1) Nos registramos en la página de Vuforia:
2) Una vez completado el registro, vamos a la esquina superior derecha y seleccionamos "*MyTrackable".*

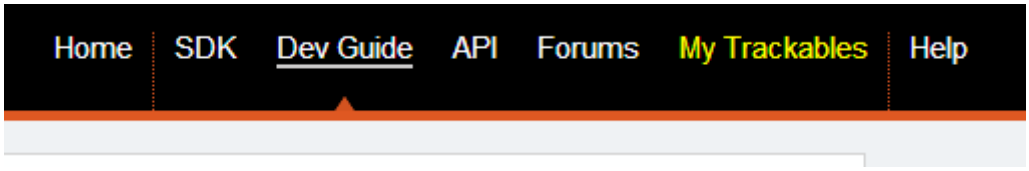


Fig.3.9.10.1 Botón "MyTrackables"

3) Pulsamos sobre "*New Project"* e indicamos un nombre para el proyecto, con el cual además se identificará el *DataSet* posteriormente en el módulo correspondiente.

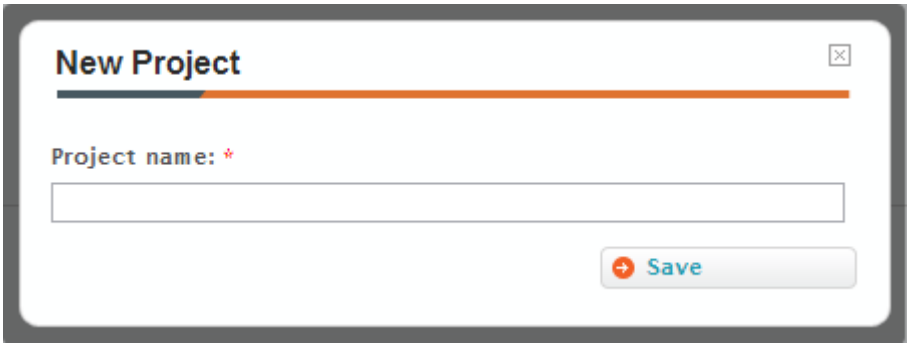


Fig.3.9.10.2 Nombre del nuevo proyecto.

4) Ahora que tenemos el proyecto creado, hemos de crear los *Trackable*o objetos a reconocer. Seleccionamos "*Create a trackable"* y luego "*Single Image"*
5) Proporcionamos un nombre para los objetos resultantes con el que identificaremos al mismo en la aplicación. Además, este tiene un máximo de 25 caracteres. Indicamos la anchura del objeto, la altura se calcula automáticamente, y finalmente pulsamos sobre el botón inferior.

Fig.3.9.10.3 Formato y tamaño del "Trackable"

6) Nos encontraremos con la pantalla de abajo, ahora solo hemos de subir nuestra propia imagen para lo que tenemos que pulsar sobre "*Upload"* y seleccionar la imagen de nuestro ordenador que queramos.

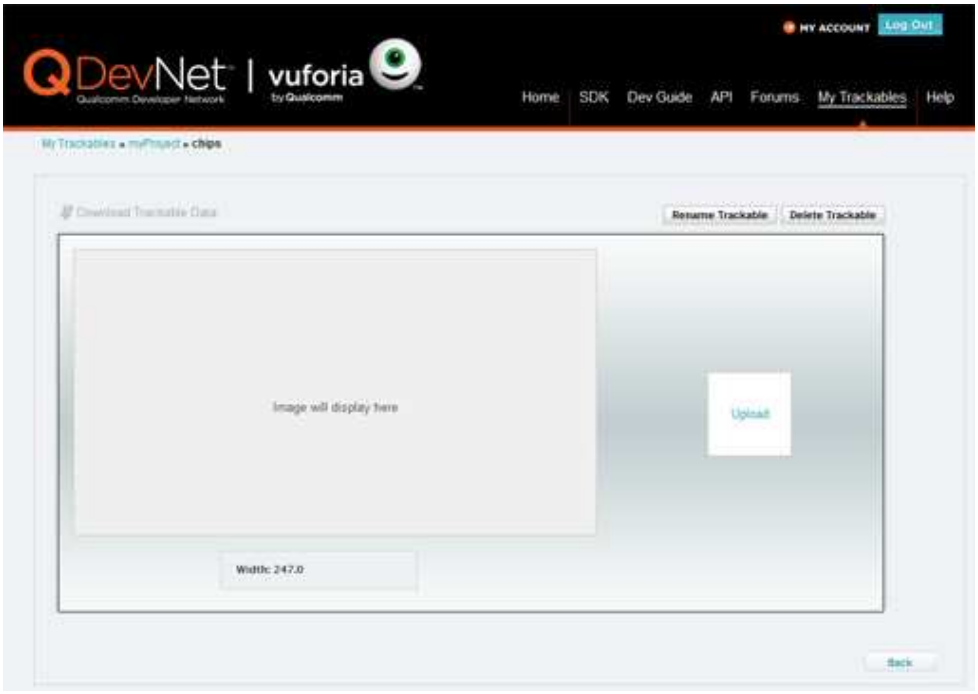


Fig.3.9.10.4 Pantalla de carga del “Trackable”

7) Tras un momento veremos nuestra imagen en blanco y negro junto a un conjunto de cruces amarillas que nos indican todos los puntos a partir de los cuales el sistema será capaz de reconocer la imagen. Además podremos ver una calificación del análisis de la imagen que nos indica si será más o menos fácil reconocerla.

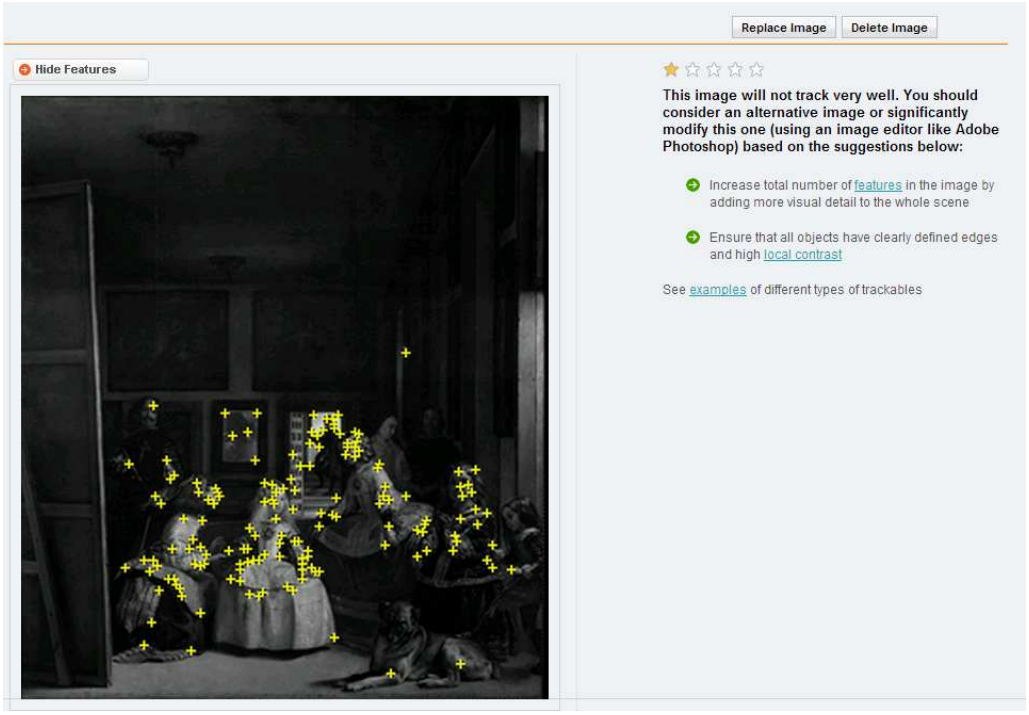


Fig.3.9.10.5 Punto de reconocimiento del “Trackable”

8) Una vez finalizado, ya podrás bajar el proyecto o *DataSet*y utilizarlo en nuestra aplicación

Fig.3.9.10.6 Selección de los "Trackables" ha descargar en el "DataSet"

9) Para descargar nuestro *DataSet* tenemos que pulsar sobre el botón "*download selected trackables"* y seleccionamos la primera opción.

Fig.3.9.10.7 Selección formato de descarga.

10) Ahora solo queda extraer el contenido en la carpeta "*<DEVELOPMENT_ROOT>\vuforia-sdk-android-xx-yy-zz\samples\Proyecto\assets"* y podemos utilizarlo en nuestro proyecto.

4. Análisis y diseño

1. Diseño Visual y Funcionamiento

En términos generales, se ha buscado el desarrollo de una aplicación de Realidad Aumentada cuyo funcionamiento esté dotado de la mayor sencillez e intuitividad posible y huyendo de las pantallas sobrecargadas de mensajes presente en tantas otras aplicaciones del estilo. Para ello, hemos dividido la aplicación en 2 partes diferenciadas, una primera parte donde veremos una representación del cuadro, enfrente al cual estamos, en blanco y negro con las partes "pulsables" en color y cuya información queremos obtener; y otra segunda parte donde proporcionaremos esa información junto al cuadro que hemos identificado y el "componente" sobre el que hemos pulsado y del que obtenemos la información.

Así pues, las aplicación presentaría el siguiente aspecto en nuestras pantallas:

1) Primero, al iniciar la aplicación, observaremos el logo de la misma sobre un fondo negro mientras los componentes y librerías son cargadas

Fig.4.1.1 Pantalla de Carga

2) A continuación nos sale simplemente la imagen captada por nuestra cámara del móvil y ya estaremos listos de apuntar sobre uno de los cuadros de nuestra base de datos.

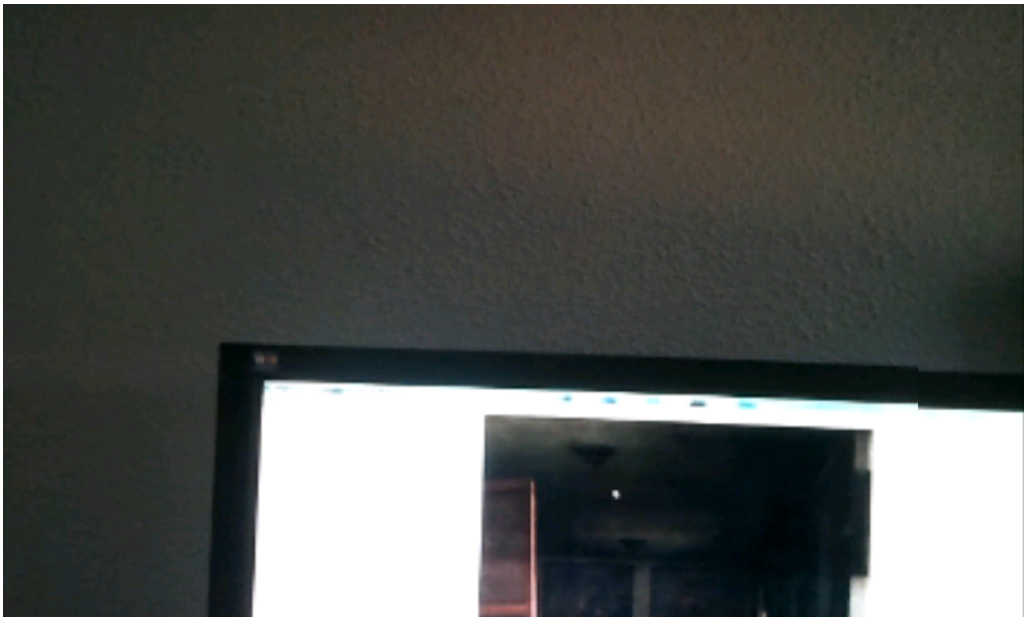


Fig. 4.1.2 Visión de la cámara en general. Se puede ver como el cuadro lo observaremos en una pantalla de ordenador, al no disponer del original.

3) Una vez enfoquemos a uno de nuestros cuadros, la aplicación lo reconocerá y si estamos a una distancia suficiente, nos representará el mismo cuadro en blanco y negro pero con los componentes “pulsables” en color para que nos sea fácil identificar cuáles son y hasta donde abarcan. Sin embargo, si donde pulsamos es la zona no coloreada, nos enlazara con la información del cuadro en general y no de alguno de sus componentes.

Fig. 4.1.3 Las partes pulsables quedan rodeadas por un borde blanco.

4) Una vez pulsado uno de los componentes, la cámara se suspende y nos encontraremos ante 5 elementos:

a) El nombre del componente del cuadro sobre el que hemos pulsado.

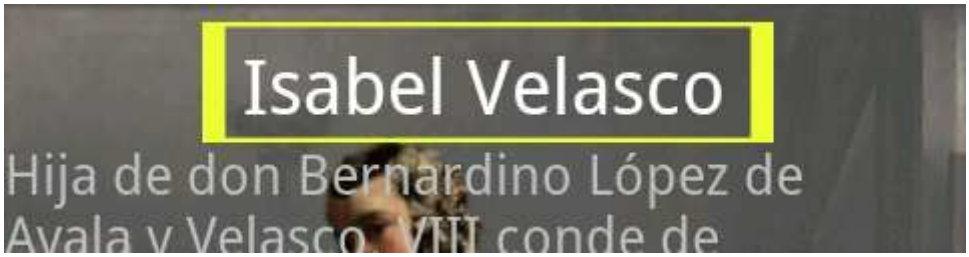


Fig. 4.1.4 Se trata del nombre del componente, Isabel Velasco

b) El texto de información sobre el componente sobre el que hemos pulsado.

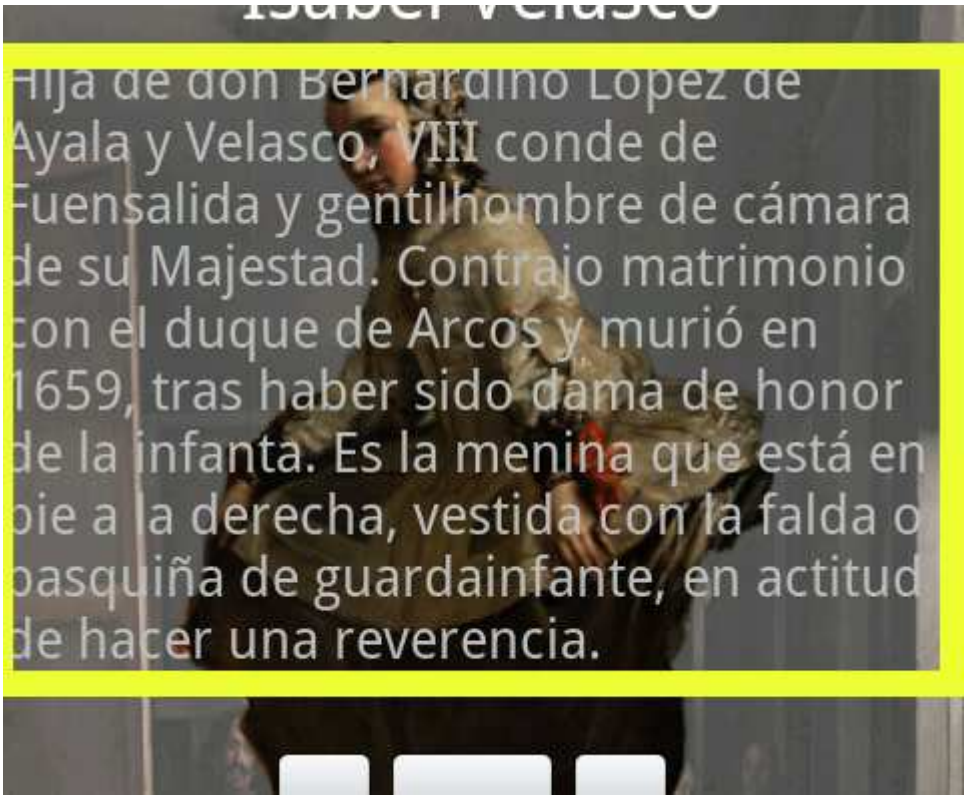


Fig. 4.1.5 La información disponible sobre Isabel Velasco

c) Detrás del texto anterior obtendremos la imagen del componente en cuestión.

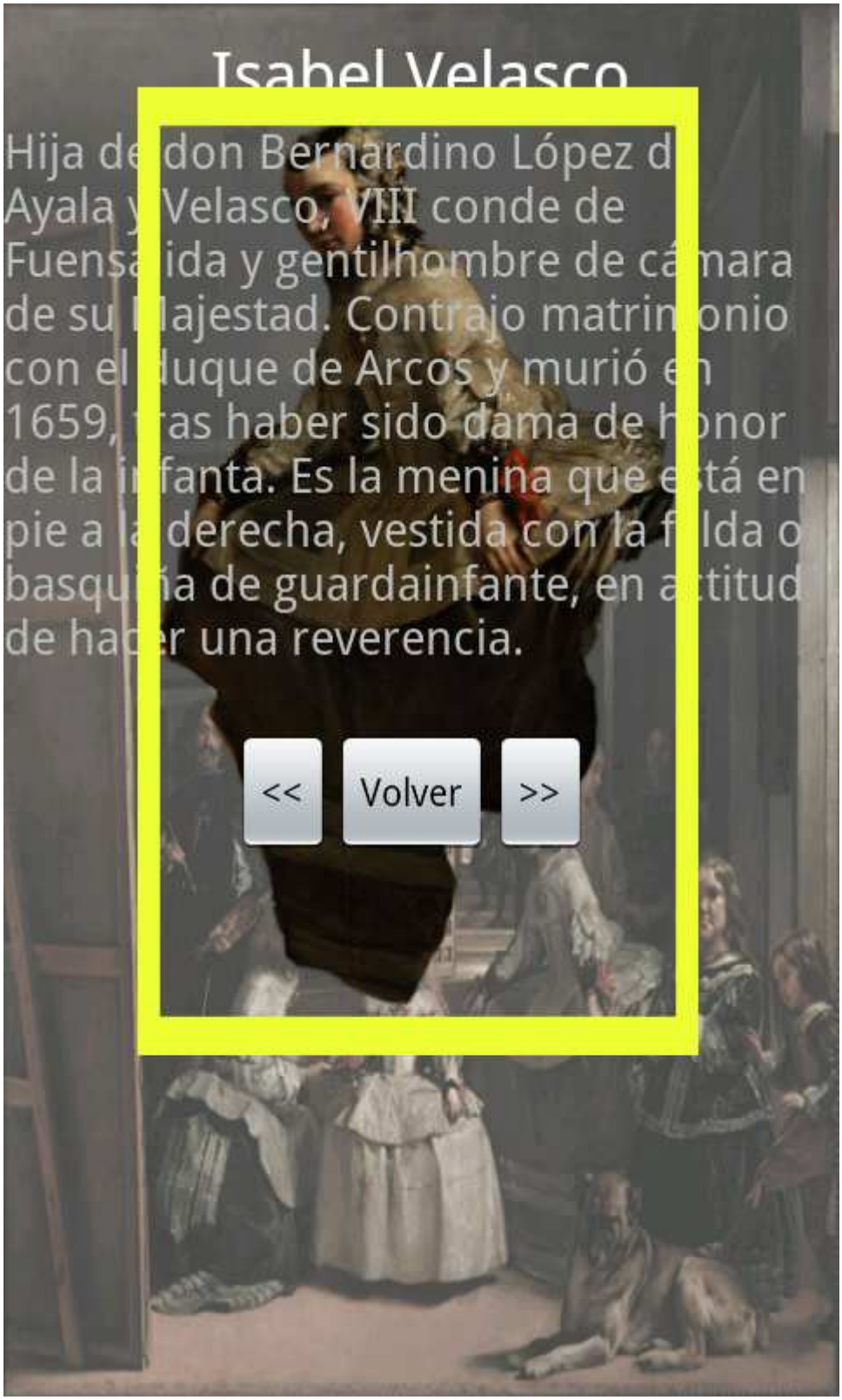


Fig. 4.1.6 Isabel Velasco como fondo.

d) Como fondo de todo lo anterior, veremos el cuadro reconocido en una tonalidad más apagada para facilitar la lectura del texto.

Fig. 4.1.7 En este caso se trata de las meninas en tonalidades apagadas.

e) 3 Botones para pasar a el componente anterior o posterior y otro más volver a la parte anterior y reconocer otro cuadro (Fig. 4.2.8). Las funcionalidades de los dos primeros también podemos realizarlas deslizando en un sentido u otro el dedo sobre la pantalla (Fig. 4.2.9).

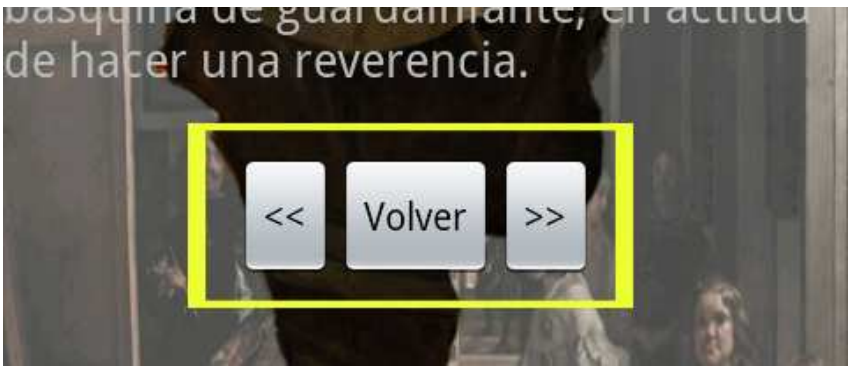


Fig.4.1.8 Botones: Anterior, Volver y Siguiente.

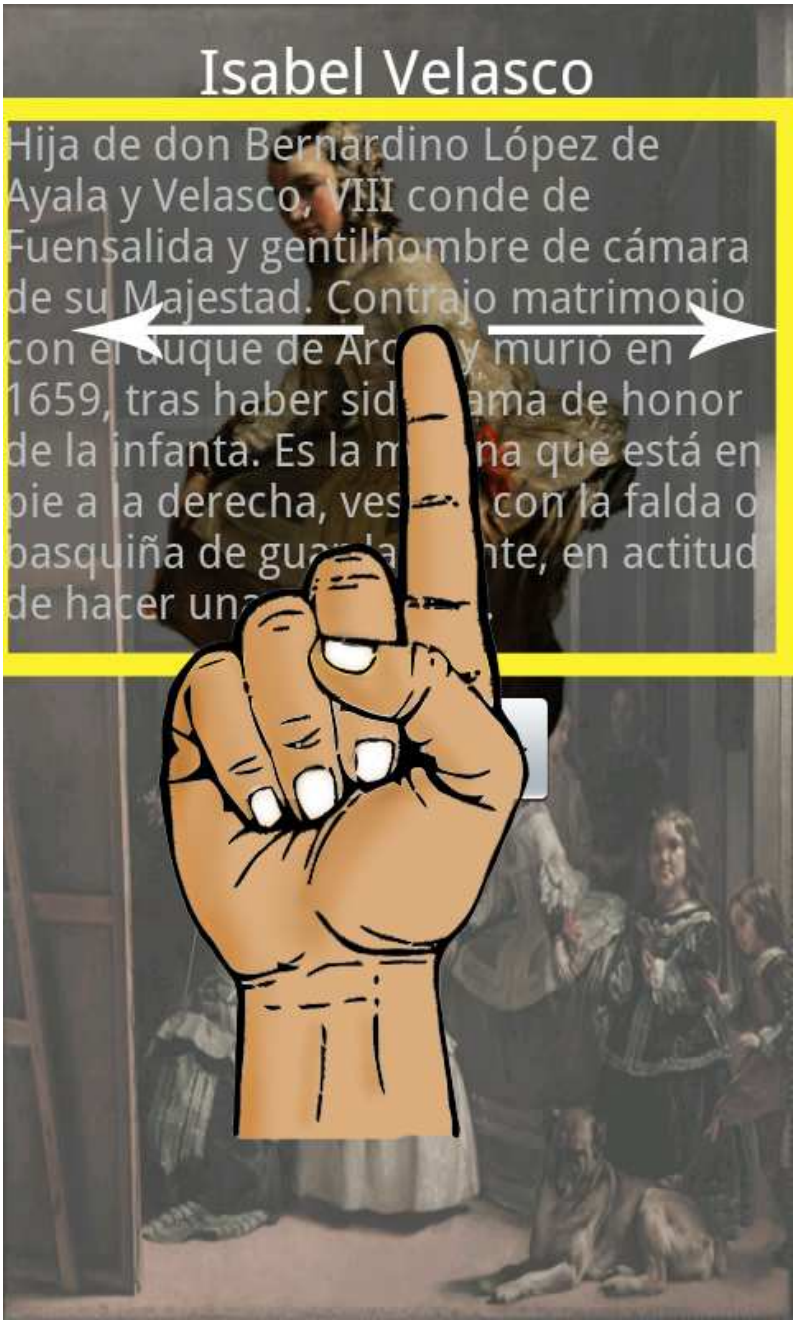


Fig 4.1.9 Anterior y Siguiente deslizando el dedo.

2. Arquitectura de Funcionamiento

Para explicar esta parte hemos de comenzar ya a diferenciar qué partes hemos desarrollado nosotros y que partes son las proporcionadas por el sistema Vuforia. Comenzaremos explicando el sistema de funcionamiento de Vuforia y a continuación explicaremos cómo hemos estructurado nuestra aplicación. De esta forma el sistema de funcionamiento de la aplicación quedaría resumido en la siguiente imagen:

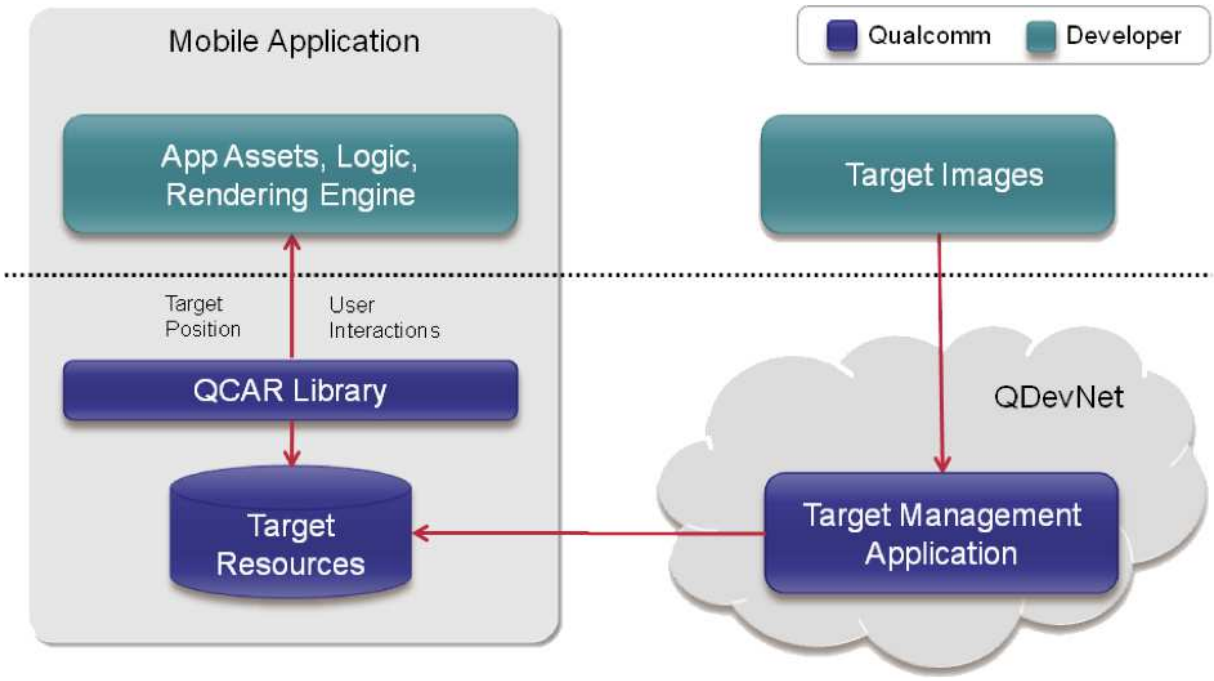


Fig. 4.2.1 Arquitectura de Vuforia

1. Arquitectura de Vuforia

Una aplicación basada en el SDK Vuforia se compone de los siguientes elementos:

- Cámara

  La cámara se asegura que cada frame es capturado y pasado eficientemente al *Tracker*. El desarrollador solo ha de decir cuando ha de comenzar y terminar de capturar. La imagen es entregada automáticamente en un formato y tamaño dependiendo del dispositivo.

- ImageConverter

  Se encarga de convertir la imagen capturada por la cámara en un formato aceptable para que OpenGL ES la renderice y trate. Esta conversión también incluye una reducción del muestreo para tener la imagen de la cámara en diferentes resoluciones para diferentes procesos.

- Tracker

  Es la parte encargada de detectar y rasterizar la imagen captada mediante algoritmos computacionales de visionado. Partiendo de la imagen captada por la cámara, se preocupa de detectar nuevos marcadores, objetos y de evaluar posibles botones virtuales. El resultado es almacenado en un objeto usado por el renderizador de video y que puede también ser accesible desde el código del desarrollador. Puede cargar distinto conjuntos de datos, patrones de reconocimiento, pero solo uno puede estar activo al mismo tiempo.

- Renderizador de Video de Fondo

  El renderizador de video de fondo realiza la tarea de analizar la imagen almacenada por la cámara en el objeto de estado. El rendimiento de la representación de vídeo de fondo está optimizado para cada dispositivo específico.

- Código de Aplicación

  El desarrollador debe inicializar todos los componentes y seguir 3 pasos claves en el código de la aplicación. Para cada frame procesado, el objeto de estado es actualizado y el método de renderizado es llamado. El código ha de hacer:

  - Preguntar al objeto de estado si ha detectado nuevos objetos, marcadores o actualizar el estado de estos elementos.
  - Actualizar la lógica de la aplicación con los nuevos datos entrantes.
  - Renderizar los gráficos correspondientes por encima de la imagen de la cámara.

- Objeto a reconocer.

  Los objetos a reconocer son creados mediante la página web con el llamado *"Target Management System".* El conjunto de datos descargados tras el proceso de creación contiene un XML de configuración que permite al desarrollador configurar algunas características del objeto y un fichero binario que contiene el patrón que permite identificar al mismo. Estos ficheros son compilados junto a la aplicación dentro de el instalador de la mismo y sin usados en tiempo de ejecución por el SDK Vuforia.

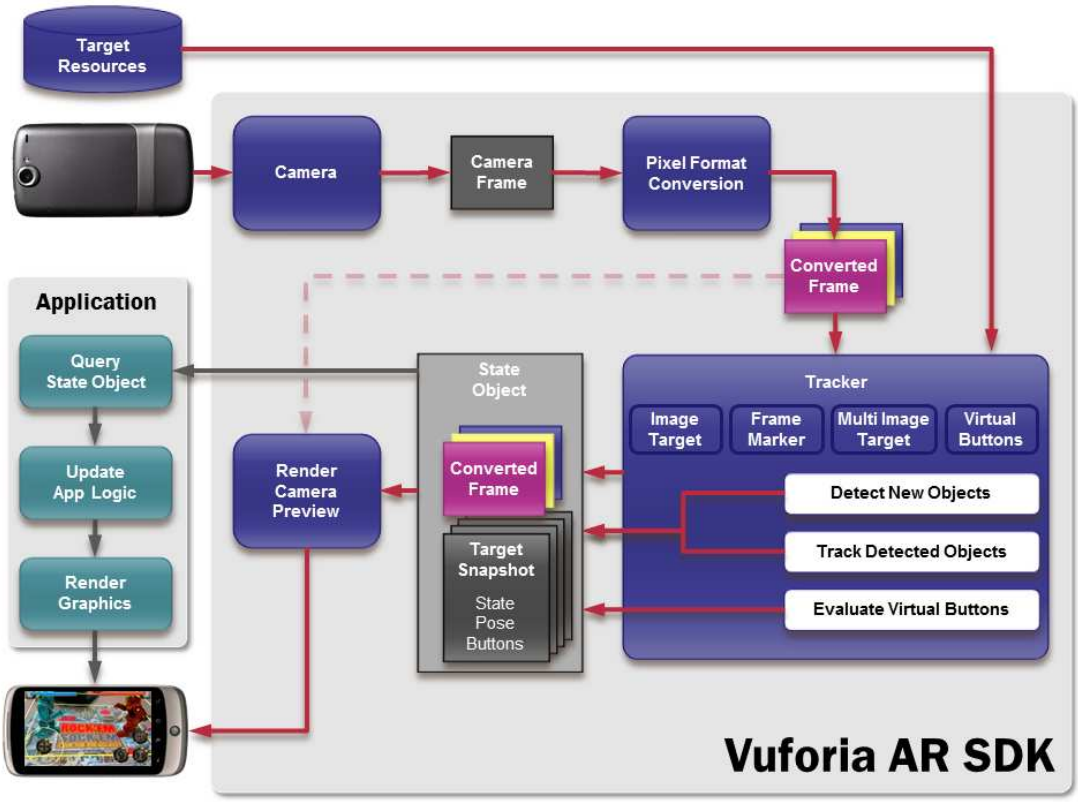


Fig. 4.2.1.1 Arquitectura SDK Vuforia

2. Arquitectura de la Aplicación

Partiendo de lo ya expuesto, hemos creado y añadido a las ya presentes, una serie de clases que responden a la identificación del patrón de unos de nuestros cuadros. Nuestra aplicación añadirá a la arquitectura de Vuforia las siguientes clases con las funciones o tareas correspondientes:

- Base de Datos

  Contiene toda la información necesaria para el funcionamiento de la aplicación. Por un lado está la información con los "carteles" que habrá que crear al identificar uno de nuestros cuadros, las coordenadas de cada punto de estos "carteles", el componente del cuadro al que ha de identificar y el nombre del mismo cuadro detectado. Por otro lado, tendremos la información sobre todos los componentes, el nombre del mismo y el del cuadro nuevamente.

- CanvasView

  Se trata del responsable de representar los "carteles" sobre los componentes del cuadro en cuestión. Además es el encargado de detectar sobre cuál de todos los componentes hemos pulsado al tocar la pantalla. Una vez detectado cual de todos eso, llama a la segunda fase de nuestra aplicación, la de muestra la información, con los datos necesarios para que esta puede sacarnos los datos correctos acerca del componente.

- Cartel

  Se trata de una clase que realiza una representación parecida a una "tapa" sobre cada uno de los componentes del cuadro. Para ello, se crea un objeto de este tipo por cada uno de ellos partiendo de las coordenadas relativas sobre el cuadro, es decir, contiene las coordenadas en proporción a la anchura y altura del cuadro de cada uno de los 4 vértices que componen el cartel. Además, son llamados por el CanvasView para detectar si la pulsación se ha hecho sobre el mismo y devuelve el valor a la misma clase.

- Info

  Es la encargada de toda la segunda fase del la aplicación. Es una Activity nueva que nos crea toda la interfaz, lee la información de la base de datos y nos permite navegar entre cada una de las entradas de la base de datos para consultar la diferente información del cuadro.

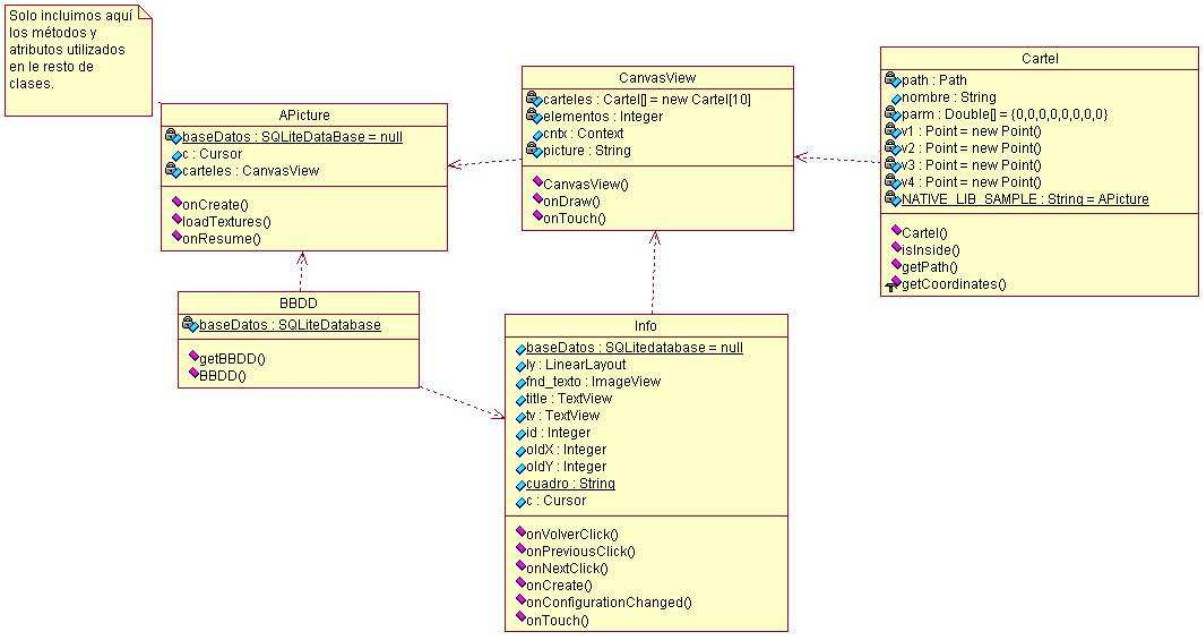


Fig. 4.2.2.1 Diagrama de Clases

3. Rotación de la pantalla

Una de las partes en las que hemos prestado atención a la hora de diseñar nuestra aplicación de RA ha sido el hecho de no tener la certeza ni la seguridad de que el usuario la utilizase de con la pantalla en un sentido u otro (vertical u horizontal). Pese a que nuestro logo de aplicación aparezca en sentido horizontal en la pantalla de carga, el usuario puede querer mantener su móvil en el sentido opuesto en todo momento. Esto nos supone que tanto el reconocimiento de imágenes como la exposición posterior de la información almacenada han de poder mostrarse en ambos sentidos y sin que suponga reiniciar o recargar la aplicación.

Para solucionar todo los problemas derivados, como el desajuste de imágenes perdiendo las proporciones de altura y anchura o que los texto aparezcan desestructurados, hemos trabajado siempre con coordenadas relativas y con las “gravedades” de los componentes, así como el uso de imágenes de tamaño fijo que son representadas en su tamaño real sin realizar ajustes dependiendo del ancho de la pantalla. Es decir, en definitiva hemos tomado las siguientes medidas:

- Todo componente de lo que se conoce como “layout” es centrado o posicionado dependiendo de su padre. De esta forma. las coordenadas de posicionamiento no son definidas de forma estática sino que se realiza un cálculo dinámico con los parámetros del padre para hallar la posición adecuada, ya sea centrado en un extremo o en el otro de la pantalla.
- Al pulsar sobre un componente del cuadro, utilizamos una serie de operaciones matemáticas que nos determinan si la pulsación ha sido dentro de uno de ellos. Esto guarda relación con la rotación ya que al girar la pantalla, el sistema de coordenadas utilizado por Android cambia y hemos de ser capaces de adaptarnos a este cambio. En posteriores apartados explicaremos mejor el sistema de captación de las pulsaciones.

- En las pantallas de información sobre los componentes mostraremos el texto adaptado al ancho de la pantalla. Con ello conseguimos proporcionar un modo de lectura más cómodo dependiendo de las preferencias de cada usuario.
- Al visualizar la información nos surgía una problema de reinicio de la Activity. Si entrabamos tras pulsar en un componente, posteriormente avanzamos o pasamos a otro componente y finalmente cambiamos de orientación, la Activity se reiniciaba con la nueva orientación y volvíamos al primer componente con el que habíamos entrado a esta segunda fase de la aplicación. Para solucionarlo, tuvimos que añadir un trozo de código para que añadir en el que indicamos que el método *onCreate* no debía ejecutarse al cambiar la orientación, que era el causante de que esto ocurriera. Posteriormente indicaremos de que código se trata.
- Como ya hemos dicho antes, las imágenes las tratamos en su tamaño original, sin adaptarse al contenido de la pantalla. Para ello, en vez de usar la imagen como fondo del campo del tipo *TextView*hemos utilizado un *ImageView* cuya anchura y altura se adapten al contenido, es decir se adapta a la propia imagen. Solo nos queda centrarla por detrás del *TextView* con el texto y ya nos hace la función que queríamos. La única excepción se trata del fondo de la pantalla de información que la hemos conservado ya que no tendría sentido hacerla *"scrollable"* y al mismo tiempo queríamos que se observase el cuadro entero como fondo, por ello los cuadros se adaptaran a la pantalla deformándose en el proceso.

5. Desarrollo

Una vez llegados a este punto, ya somos capaces de ponernos a trabajar en el desarrollo del proyecto. Para ello, antes de intentar directamente la construcción de la aplicación hemos de seguir un proceso o etapa de aprendizaje tanto del sistema operativo Android, la programación en Java o la comunicación en código nativo entre C++ y Java. Además, el desarrollo de la aplicación y el progreso en la construcción de esta se vio afectada por una serie de problemas y contratiempos que dificulta su implementación hasta pasados varios meses del comienzo del proyecto y que se explicamos en el apartado correspondiente más adelante.

Así, el proyecto lo hemos desarrollado en las siguientes fases:

- Aprendizaje del SO Android
- Estudio de la arquitectura del SDK Vuforia
- Desarrollo de la Aplicación
- Diseño de la Aplicación
- Implementación de código
- Prueba de Funcionamiento
- Escritura de la Memoria del proyecto

1. Compilación y Ejecución de los proyectos Vuforia

Una vez tenemos todos los componentes necesarios para la creación de nuestro proyecto de RA con Vuforia, estos son los pasos que hemos de seguir para la compilación y ejecución del mismo.

1) Compilamos las librerías compartidas: para ello hemos de navegar mediante la herramienta Cygwin hasta la ruta raíz de nuestro proyecto:

   <DEVELOPMENT_ROOT>\vuforia-sdk-android-xx-yy-zz\samples\APicture

2) Ejecutamos el comando *ndk-build*y ha de responder la terminal con algo parecido a lo siguiente:

```
ndk-build

Gdbserver      : [arm-linux-androideabi-4.4.3] libs/armeabi/gdbserver
Gdbsetup       : libs/armeabi/gdb.setup
Gdbserver      : [arm-linux-androideabi-4.4.3] libs/armeabi-v7a/gdbserver
Gdbsetup       : libs/armeabi-v7a/gdb.setup
Compile++ arm  : APicture<= APicture.cpp
Compile++ arm  : APicture<= SampleUtils.cpp
Compile++ arm  : APicture<= Texture.cpp
StaticLibrary  : libstdc++.a
Prebuilt       : libQCAR.so <= jni/../../../build/lib/armeabi/
```

```
SharedLibrary  : libAPicture.so
Install        : libAPicture.so => libs/armeabi/libAPicture.so
Install        : libQCAR.so => libs/armeabi/libQCAR.so
Compile++ arm  : APicture<= APicture.cpp
Compile++ arm  : APicture<= SampleUtils.cpp
Compile++ arm  : APicture<= Texture.cpp
StaticLibrary  : libstdc++.a
Prebuilt       : libQCAR.so <= jni/../../../build/lib/armeabi-v7a/
SharedLibrary  : libImageTargets.so
Install        : libAPicture.so => libs/armeabi-v7a/libAPicture.so
Install        : libQCAR.so => libs/armeabi-v7a/libQCAR.so
```

3) Creación del proyecto en el entorno de desarrollo Eclipse. Para ello seguimos los pasos:
   a) Abrimos el IDE Eclipse.
   b) En el margen superior izquierdo, seleccionamos *File->New->Proyect*
   c) En el menú que nos aparece elegimos *Android->Android Proyect*

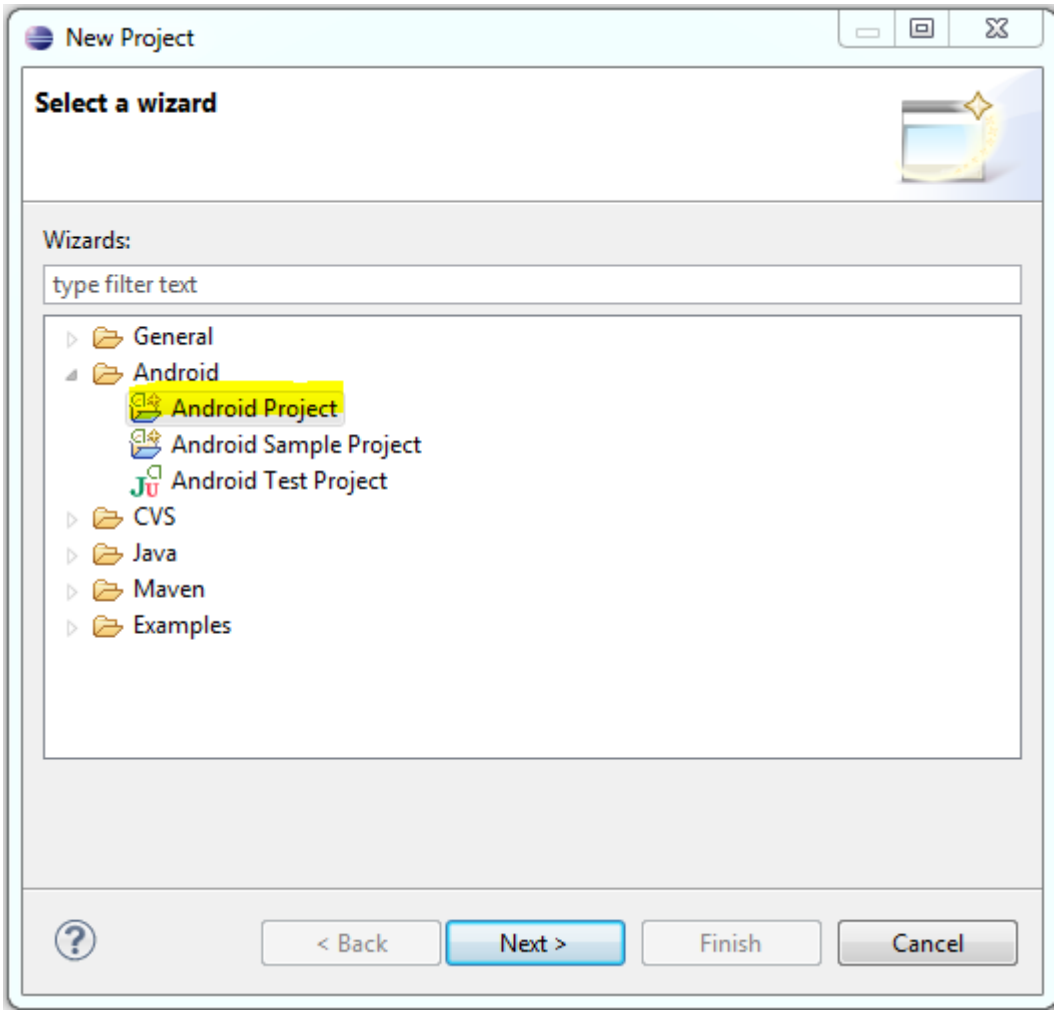


Fig. 5.1.1 Crear proyecto Android nuevo

d) Seleccionamos la opción *"Create proyect from existing source"* y en la ruta ponemos haya donde tengamos el proyecto Ejemplo de Vuforia.

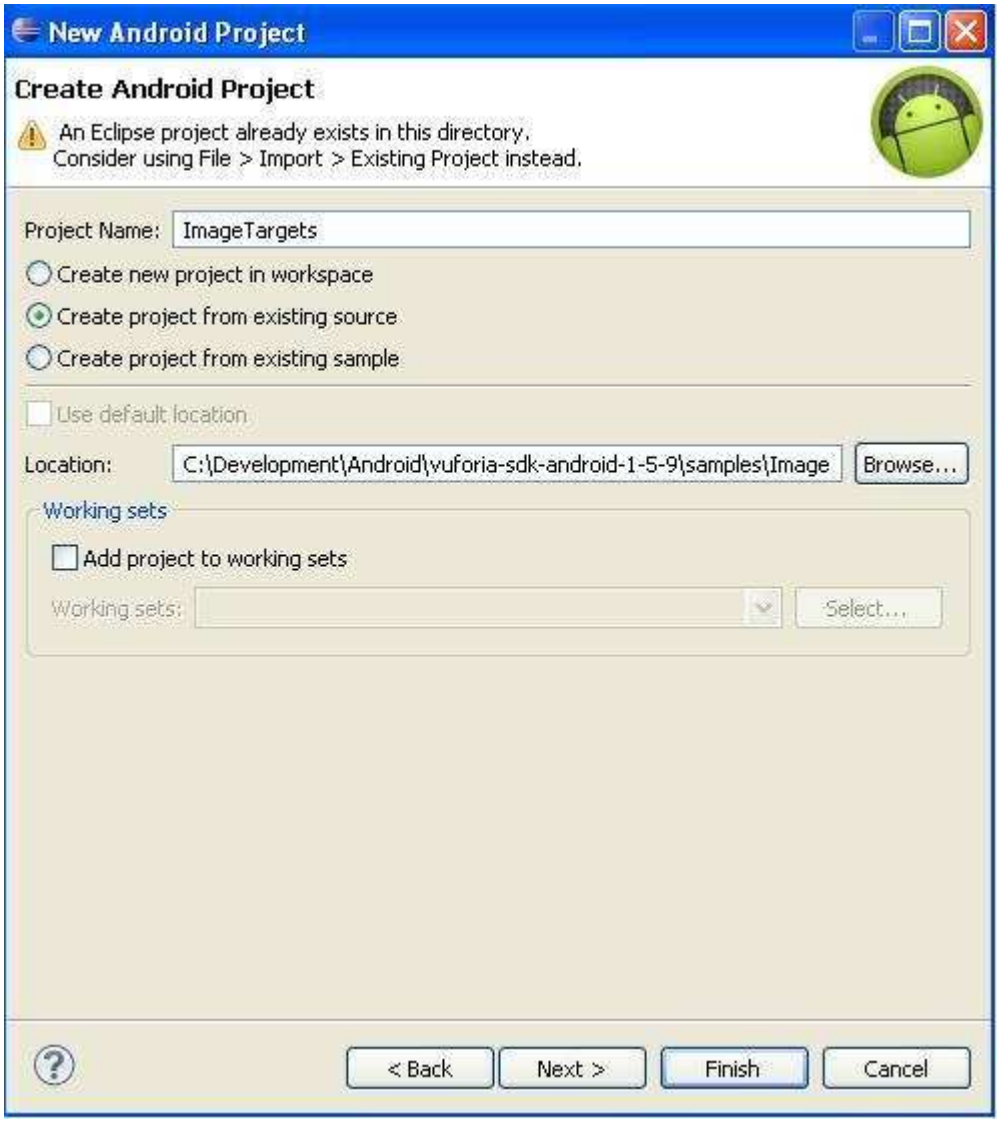


Fig. 5.1.2 Crear proyecto desde fuente existente.

Nosotros nos dedicaremos a modificar este ejemplo y posteriormente guardarlo con el nombre que queramos en nuestra carpeta. Toda esta transición se explicará en siguientes apartados.

e) Pulsamos el botón Finish y ya tendremos las carpetas y organización del los ficheros disponible en la parte izquierda de nuestro espacio de trabajo.
f) Llegados a este punto, suponemos que ya hemos modificado y creado nuestra aplicación (explicada en el siguiente apartado) y procedemos a la ejecución de nuestro proyecto. Para ello solo hemos de pulsar sobre *Run->Run As->Android.*
g) Ahora solo nos queda ponernos frente a una imagen del cuadro que queramos, ya sea impresa o directamente sobre el original, y apuntar con la cámara.

2. Fases de Desarrollo

1. Aprendizaje del SO Android

El alumno proyectista parte de una base de conocimiento de programación en el lenguaje Java pero desconoce el funcionamiento y las peculiaridades del sistema operativo Android. Por ello, fue preciso dedicar un tiempo a su estudio.

Con este fin, se decidimos realizar una búsqueda de diversos tutoriales, tanto videotutoriales como escritos, de aprendizaje de programación en Android, entre los que cabe destacar los videotutoriales de *edu4java*. No resultó una tarea complicada ya que hoy en día está muy de moda la programación para dispositivos móviles y abundan tanto libros de introducción como páginas en internet dedicadas, al menos en parte, a la enseñanza de este tipo de programación.

En esta fase se aprendió como realizar algunas de las cosas básicas como son:

- Componentes y Estructuras de un programa Android
- Elementos y estructura de la interfaz de usuario
- Manejo de Intent para ejecución de nuevas Activity's
- Manejo de SQLite como base de datos

Con todo lo aprendimos, realizamos nuestros primeros programas en el sistema operativo y una vez nos aclaramos de forma general cómo habíamos de construir una aplicación para Android, empezamos a realizar pruebas con aquello que consideramos interesante o necesario conocer más afondo para la hora de implementar nuestra propia aplicación, como es la detección de las pulsaciones en pantalla, superponer diferentes componentes de la interfaz de usuario o como realizar el paso de parámetros entre diferentes activity's.

Una vez adquirimos el conocimiento suficiente para trabajar con todos y cada uno de estos elementos, podemos pasar a la siguiente fase en la que investigaremos el funcionamiento de la ya comentada arquitectura de Vuforia y como se comunican y trabajan los módulos de esta.

2. Estudio de la Arquitectura del SDK Vuforia

Una vez tenemos el conocimiento básico de programación en Android, pasamos a investigar cómo funciona y se organizan los módulos del SDK de Vuforia. Realizamos un estudio del código de todos los módulos identificando las partes relevantes para modificar el ejemplo suministrado por el SDK y que realice la tarea que queramos. De tal forma que sacamos las siguientes parte organizadas por módulos:

ImageTargets.java

```
protected void onCreate(Bundle savedInstanceState)
{
    DebugLog.LOGD("ImageTargets::onCreate");
    super.onCreate(savedInstanceState);

    // Set the splash screen image to display during initialization:
    mSplashScreenImageResource = R.drawable.splash_screen_image_targets;

    // Load any sample specific textures:
    mTextures = new Vector<Texture>();
    loadTextures();

    // Query the QCAR initialization flags:
    mQCARFlags = getInitializationFlags();

    // Update the application status to start initializing application
    updateApplicationStatus(APPSTATUS_INIT_APP);
}
```

Fig. 5.2.2.1 onCreate()

Al ejecutar la aplicación, es esta parte del código la que se encarga de asignar la imagen de carga, la que vemos al inicio de la aplicación y que nosotros modificaremos por un icono propio.

```
/** We want to load specific textures from the APK, which we will later
use for rendering. */
private void loadTextures()
{
    mTextures.add(Texture.loadTextureFromApk("TextureTeapotBrass.png",
                                             getAssets()));
    mTextures.add(Texture.loadTextureFromApk("TextureTeapotBlue.png",
                                             getAssets()));
    mTextures.add(Texture.loadTextureFromApk("TextureTeapotRed.png",
            getAssets()));
}
```

Fig. 5.2.2.2 loadTextures()

Una vez es reconocido el objeto que queremos reconocer se procede a renderizar un objeto 3d, en este caso una tetera, y a esa tetera se le aplica una textura. Esta es la parte del código que define qué texturas son las aplicables y las almacena de forma ordenada para que posteriormente sean accesibles y aplicables.

```java
/** Called when the activity will start interacting with the user.*/
 protected void onResume()
 {
     DebugLog.LOGD("ImageTargets::onResume");
     super.onResume();

     // QCAR-specific resume operation
     QCAR.onResume();

     // We may start the camera only if the QCAR SDK has already been
     // initialized
     if (mAppStatus == APPSTATUS_CAMERA_STOPPED)
     {
         updateApplicationStatus(APPSTATUS_CAMERA_RUNNING);

         // Reactivate flash if it was active before pausing the app
         if (mFlash)
         {
             boolean result = activateFlash(mFlash);
             DebugLog.LOGI("Turning flash "+(mFlash?"ON":"OFF")+" "+(result?"WORKED":"FAILED")+"!!");
         }
     }

     // Resume the GL view:
     if (mGlView != null)
     {
         mGlView.setVisibility(View.VISIBLE);
         mGlView.onResume();
     }
 }
```

Fig. 5.2.2.3 onResume()

Es llamado cuando la actividad comienza a interactuar con el usuario. Aquí es donde debemos ponernos a la escucha de posibles mensajes de otro de los módulos diciéndonos que ha reconocido uno de los objetos (cuadros).

ImageTargetsRenderer.java

Como el nombre indica, es el módulo encargado de llamar a la renderización de los objetos que correspondan en cada frame. Pero en nuestro caso, tras consultar y preguntar cómo realizar una serie de tareas le hemos dado una función de paso intermedio entre los módulos homónimos en lenguaje C++ y Java. En el desarrollo de nuestra aplicación no modificaremos nada, solo añadiremos código adicional.

ImageTargets.cpp

Se trata del módulo principal de cara al funcionamiento del sistema Vuforia de Realidad Aumentada. En él, encontramos una serie de funciones del JNI, Java Native Interface, es decir, un conjunto de funciones que nos permiten la comunicación de objetos en código C++ con Java. Estos métodos en código nativo, manejan todos los aspectos del sistema, desde el control de la cámara hasta el reconocimiento de los patrones y lo más importante, y donde hemos trabajado, se encarga de realizar la construcción, que no representación, de los objetos a sacar al reconocer uno de los patrones.

Tras el estudio del código, los siguientes métodos y líneas son las que hemos creído más relevantes de cara a nuestro proyecto:

```
// Constants:
static const float kObjectScale = 3.f;

QCAR::DataSet* dataSetStonesAndChips    = 0;
QCAR::DataSet* dataSetTarmac            = 0;

bool switchDataSetAsap          = false;
```

Fig. 5.2.2.4 Identificación de los “DataSets”

Situado al comienzo del código, se trata de unas constantes que nos definen la escala con la que se representamos el objeto 3D, en este caso la tetera. Los *DataSet* son constantes que nos definirán cuál de los conjuntos de datos (patrones de reconocimiento) está activo, por lo que se irán turnando activándose cuando el otro se desactive. Y por último, se trata de un controlador de que nos indica si debemos cambiar de conjunto de datos. Esto es necesario ya que el número de patrones reconocibles está limitado por cada *DataSet*y por lo tanto sería necesario cambiar, por ejemplo en nuestro proyecto, al cambiar de museo.

```
// Object to receive update callbacks from QCAR SDK
class ImageTargets_UpdateCallback : public QCAR::UpdateCallback
{
    virtual void QCAR_onUpdate(QCAR::State& /*state*/)
    {
        if (switchDataSetAsap)
        {
            switchDataSetAsap = false;

            // Get the image tracker:
            QCAR::TrackerManager& trackerManager = QCAR::TrackerManager::getInstance();
            QCAR::ImageTracker* imageTracker = static_cast<QCAR::ImageTracker*>(
                trackerManager.getTracker(QCAR::Tracker::IMAGE_TRACKER));
            if (imageTracker == 0 || dataSetStonesAndChips == 0 || dataSetTarmac == 0 ||
                imageTracker->getActiveDataSet() == 0)
            {
                LOG("Failed to switch data set.");
                return;
            }

            if (imageTracker->getActiveDataSet() == dataSetStonesAndChips)
            {
                imageTracker->deactivateDataSet(dataSetStonesAndChips);
                imageTracker->activateDataSet(dataSetTarmac);
            }
            else
            {
                imageTracker->deactivateDataSet(dataSetTarmac);
                imageTracker->activateDataSet(dataSetStonesAndChips);
            }
        }
    }
};
```

Fig. 5.2.2.5 Cambio entre “DataSets”

En esta parte del código estamos realizando el cambio entre cada uno de los *DataSet* de forma que nunca puedan estar activos dos al mismo tiempo, ya que el sistema no es capaz de manejarse con más de uno al mismo tiempo.

```
JNIEXPORT int JNICALL
Java_com_qualcomm_QCARSamples_ImageTargets_ImageTargets_loadTrackerData(JNIEnv *, jobject)
{
    .
    .
    .
    // Create the data sets:
    dataSetStonesAndChips = imageTracker->createDataSet();
    if (dataSetStonesAndChips == 0)
    {
        LOG("Failed to create a new tracking data.");
        return 0;
    }
    .
    .
    .
    // Load the data sets:
    if (!dataSetStonesAndChips->load("StonesAndChips.xml", QCAR::DataSet::STORAGE_APPRESOURCE))
    {
        LOG("Failed to load data set.");
        return 0;
    }
    .
    .
    .
    // Activate the data set:
    if (!imageTracker->activateDataSet(dataSetStonesAndChips))
    {
        LOG("Failed to activate data set.");
        return 0;
    }
}
```

Fig. 5.2.2.6 Crear, Cargar y Activar los “DataSets”

Lo que vemos en la imagen se trata de una versión reducida de todo el método en el que podemos ver como se crean, cargan y activan los *DataSet.* Es en esta parte del método donde definiremos qué archivo XML es el que nos define nuestros cuadros, para ser precisos en la parte del *Load* como puede verse en la imagen en gris claro. El mismo proceso se haría con el resto de *DataSet* ahí donde se ven 3 puntos suspensivos verticalmente.

```
JNIEXPORT void JNICALL
Java_com_qualcomm_QCARSamples_ImageTargets_ImageTargetsRenderer_renderFrame(JNIEnv *, jobject)
{
    .
    .
    .

    // Did we find any trackables this frame?
    for(int tIdx = 0; tIdx < state.getNumActiveTrackables(); tIdx++)
    {
        // Get the trackable:
        const QCAR::Trackable* trackable = state.getActiveTrackable(tIdx);
        QCAR::Matrix44F modelViewMatrix =
            QCAR::Tool::convertPose2GLMatrix(trackable->getPose());

        // Choose the texture based on the target name:
        int textureIndex;
        if (strcmp(trackable->getName(), "chips") == 0)
        {
            textureIndex = 0;
        }
        else if (strcmp(trackable->getName(), "stones") == 0)
        {
            textureIndex = 1;
        }
        else
        {
            textureIndex = 2;
        }

    .
    .
    .
```

Fig. 5.2.2.7 Comprobar si hemos encontrado un “Trackable” y cargar la textura que convenga

Se trata del comienzo del método más importante del sistema, aquí realizamos las acciones que queremos que sucedan al reconocer uno de los objetos (*trackable)*. Tras los códigos suspensivos que nos representa código no relevante para la explicación, podemos observar como recorremos la lista de objetos reconocibles cargándolos y dependiendo de cuál de ellos es, le asignamos una textura, la cual habíamos cargado en el ImageTargets.java y ahora solo es un número de una lista.

```
#ifdef USE_OPENGL_ES_1_1
        // Load projection matrix:
        glMatrixMode(GL_PROJECTION);
        glLoadMatrixf(projectionMatrix.data);

        // Load model view matrix:
        glMatrixMode(GL_MODELVIEW);
        glLoadMatrixf(modelViewMatrix.data);
        glTranslatef(0.f, 0.f, kObjectScale);
        glScalef(kObjectScale, kObjectScale, kObjectScale);

        // Draw object:
        glBindTexture(GL_TEXTURE_2D, thisTexture->mTextureID);
        glTexCoordPointer(2, GL_FLOAT, 0, (const GLvoid*) &teapotTexCoords[0]);
        glVertexPointer(3, GL_FLOAT, 0, (const GLvoid*) &teapotVertices[0]);
        glNormalPointer(GL_FLOAT, 0,  (const GLvoid*) &teapotNormals[0]);
        glDrawElements(GL_TRIANGLES, NUM_TEAPOT_OBJECT_INDEX, GL_UNSIGNED_SHORT,
                       (const GLvoid*) &teapotIndices[0]);
#else

        QCAR::Matrix44F modelViewProjection;

        SampleUtils::translatePoseMatrix(0.0f, 0.0f, kObjectScale,
                                         &modelViewMatrix.data[0]);
        SampleUtils::scalePoseMatrix(kObjectScale, kObjectScale, kObjectScale,
                                     &modelViewMatrix.data[0]);
        SampleUtils::multiplyMatrix(&projectionMatrix.data[0],
                                    &modelViewMatrix.data[0] ,
                                    &modelViewProjection.data[0]);

        glUseProgram(shaderProgramID);

        glVertexAttribPointer(vertexHandle, 3, GL_FLOAT, GL_FALSE, 0,
                              (const GLvoid*) &teapotVertices[0]);
        glVertexAttribPointer(normalHandle, 3, GL_FLOAT, GL_FALSE, 0,
                              (const GLvoid*) &teapotNormals[0]);
        glVertexAttribPointer(textureCoordHandle, 2, GL_FLOAT, GL_FALSE, 0,
                              (const GLvoid*) &teapotTexCoords[0]);

        glEnableVertexAttribArray(vertexHandle);
        glEnableVertexAttribArray(normalHandle);
        glEnableVertexAttribArray(textureCoordHandle);

        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, thisTexture->mTextureID);
        glUniformMatrix4fv(mvpMatrixHandle, 1, GL_FALSE,
                           (GLfloat*)&modelViewProjection.data[0] );
        glDrawElements(GL_TRIANGLES, NUM_TEAPOT_OBJECT_INDEX, GL_UNSIGNED_SHORT,
                       (const GLvoid*) &teapotIndices[0]);
```

Fig. 5.2.2.8 Código de construcción de Objetos 3D

Por último, este es el código que nos construye el objeto 3D en OpenGL ES a partir de una serie de parámetros y cabeceras contenidas en una librería importada al comienzo del módulo.

3. Desarrollo de la Aplicación

Llegados a esta fase del desarrollo del proyecto, ya conocemos el sistema de funcionamiento y la arquitectura de Vuforia, el API de Qualcomm sobre el que vamos a desarrollar nuestra la aplicación. Sabemos cómo crear los llamados *Trackables* o imágenes reconocibles por el sistema y como añadirlos al mismo. Tenemos pues ya la

capacidad de construir la primera parte de la aplicación en la que hemos de reconocer la imagen del cuadro que queramos y al pulsar interaccione y nos lleve como respuesta a la segunda parte de la aplicación.

Además, ya somos capaces de realizar aplicaciones para el sistema operativo Android trabajando con bases de datos, interfaz de usuario o el envío de mensajes entre Activity's por ejemplo. En definitiva tenemos el conocimiento básico de programación en Android y XML. Todo ello nos permite en este punto realizar la segunda parte de nuestra aplicación, en la que exponemos la información almacenada sobre el cuadro en cuestión.

Para el desarrollo del proyecto seguiremos el modelo evolutivo en el que realizaremos sucesivas versiones de la aplicación añadiendo, modificando o quitando elementos de la misma hasta que obtengamos un resultado satisfactorio y acorde a los objetivos perseguidos. Para ello realizaremos una serie de iteraciones en espiral en las que primero diseñaremos que hemos de conseguir en la "vuelta" correspondiente.

Primera Iteración de desarrollo

Una vez tenemos el conocimiento necesario para implementar el código de la aplicación y sobre todo que es lo que podemos y no podemos hacer de forma aceptable, nos disponemos a realizar un primer diseño, que no el definitivo, sobre la aplicación. En las primeras fases del proyecto en las que creamos las estructuras básicas de funcionamiento de nuestra aplicación, trabajaremos con un único componente, la Infanta, del cuadro de Las Meninas para no enredarnos con demasiados datos y componentes. Del mismo modo, al comienzo solo tenemos reconocible un cuadro, el de Las Meninas, y posteriores iteraciones del desarrollo añadiremos más cuadros.

Así, pensamos un primer diseño de nuestro programa de RA en el que los objetos consistirán en representaciones 3d de los componentes del cuadro que al ser pulsados habrán de responder con una pantalla de información acerca del respectivo cuadro y elemento del mismo. Además, diseñamos la interfaz de usuario de la pantalla de información en la que obtendremos, en un primer momento, una pantalla con Título, Texto y fondo; así como la capacidad de navegar entre los elementos del cuadro deslizando el dedo sobre la pantalla como en tantas otras aplicaciones de lectura de texto y páginas.

A la hora de implementarlo, nos damos cuenta tanto de la complejidad que supone realizar un modelo 3D de cada elemento del cuadro identificable, su texturización completa como el peso que supondrá para la aplicación cargar todos esos modelos. Por ello, decidimos en futuras versiones diseñar otro modo de interacción y representación de los objetos con el usuario. Pese a ello, en esta primera fase proseguimos con el resto del diseño y realizamos la pantalla de información. En un primer momento, debido a los riesgos de trabajar con bases de datos y puesto que estamos comenzando, almacenaremos en un fichero de texto todos los datos(nombre del cuadro, nombre de componente, información...) al que accederemos desde nuestra pantalla de información. De esta forma, obtenemos una pantalla similar a la final

(Fig4.2.6) exceptuando la presencia de los botones inferiores y la imagen del componente en el que nos encontramos.

Hemos creado pues una clase java con las siguientes partes:

- onCreate(): se ejecuta al instanciar la Activity y es la encargada de crear todo el interfaz y componentes de la misma. En este punto, hemos supuesto que junto a la llamada a la nueva Activity nos llegan una serie de parámetros que nos permiten identificar y mostrar los componentes precisos que deseamos y no otros.

```
public void onCreate(Bundle savedInstanceState){
    Animation animacion = AnimationUtils.loadAnimation(this, R.anim.slide_left);

    super.onCreate(savedInstanceState);
    setContentView(R.layout.inf); //lee xml y lo combierte en la pantalla

    //Recogemos las variables
    Bundle bundle = getIntent().getExtras();
    cuadro = bundle.getString("cuadro");
    String componente = bundle.getString("componente");

    //Ponemos el fondo que corresponde
    ly = (LinearLayout) findViewById(R.id.fondo);
    int bckg = getResources().getIdentifier("fnd_"+cuadro, "drawable",getPackageName());
    ly.setBackgroundResource(bckg);
    ly.startAnimation(animacion);

    //Ahora solo queda rellenar el texto. Primero recogeremso el Id del textview
    title = (TextView) findViewById(R.id.titulo);
    title.setText(componente);
    tv = (TextView) findViewById(R.id.texto);

    //Asignamos la imagen correspondiente
    int img = getResources().getIdentifier(componente, "drawable",getPackageName());
    tv.setBackgroundResource(img);
    tv.setOnTouchListener(this);
```

Fig. 5.2.3.1.1 OnCreate() info.java

- onTouch(): se trata del método que captura el evento producido al interactuar el usuario con la pantalla y define la respuesta ante el mismo. En el caso, al detectar que entre que se pulsa la pantalla y se levanta el dedo hay una diferencia en uno u otro sentido en el eje X, identificamos que ha de pasar al siguiente o anterior componente almacenado en el fichero que nos hace de BBDD provisional. Una vez tenemos los datos en las variables correctas, las asignamos a los componentes del interfaz. El código de paso al siguiente/anterior componente se ha omitido debido a que en versiones posteriores se ha cambiado para que trabaje con BBDD. Además, la implementación de la animación durante la transición entre los elementos presentes en el fichero (los componentes de cuadro) se realiza en la siguiente iteración, en la presente solo nos preocupamos de que esta se realice sin problemas.

```
public boolean onTouch(View v, MotionEvent event){
    Animation animL = AnimationUtils.loadAnimation(this, R.anim.slide_left);
    Animation animR = AnimationUtils.loadAnimation(this, R.anim.slide_right);
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            oldX = (int) event.getX();
            oldY = (int) event.getY();
            break;
        case MotionEvent.ACTION_UP:
            if ( (oldX > (int) event.getX()) ){
                //Movimiento de izq a drch
                ly.startAnimation(animL);
                try{
                    //SIGUIENTE COMPONENTE DEL FICHERO

                    //Ponemos el titulo
                    title.setText(titulo);

                    tv.setText(texto);
                }catch(Exception ex){
                    DebugLog.LOGD("Excepcion1: "+ex.toString());
                    title.setText("ERROR1");
                    tv.setText("ERROR1");
                }
            }else if((oldX < (int) event.getX()) ){
                //Movimiento de drch a izq
                ly.startAnimation(animR);
                try{
                    //ANTERIOR COMPONENTE DEL FICHERO

                    //Ponemos el titulo
                    title.setText(titulo);

                    tv.setText(texto);
                }catch(Exception ex){
                    DebugLog.LOGD("Excepcion2: "+ex.toString());
                    title.setText("ERROR2");
                    tv.setText("ERROR2");
                }
            }
            break;
    }
    return true;
```

Fig. 5.2.3.1.2 OnTouch() info.java

El fichero XML del Layout del interfaz de la parte de información quedaría:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fondo"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_gravity="bottom|center_horizontal"
    android:baselineAligned="true"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/titulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:paddingTop="10sp"
        android:text="Titulo"
        android:textSize="25sp" />

    <ScrollView
        android:visibility="visible"
        android:id="@+id/scrollView1"
        android:layout_width="match_parent"
        android:keepScreenOn="true"
        android:scrollbarStyle="insideInset"
        android:layout_height="wrap_content" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical"
            android:isScrollContainer="true"
            android:scrollbars="vertical"  >

        <TextView
            android:id="@+id/texto"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

            android:paddingBottom="25sp" />
        </LinearLayout>

    </ScrollView>

</LinearLayout>
```

Fig. 5.2.3.1.3 info.xml

Dando como resultado una pantalla esquemáticamente del estilo:

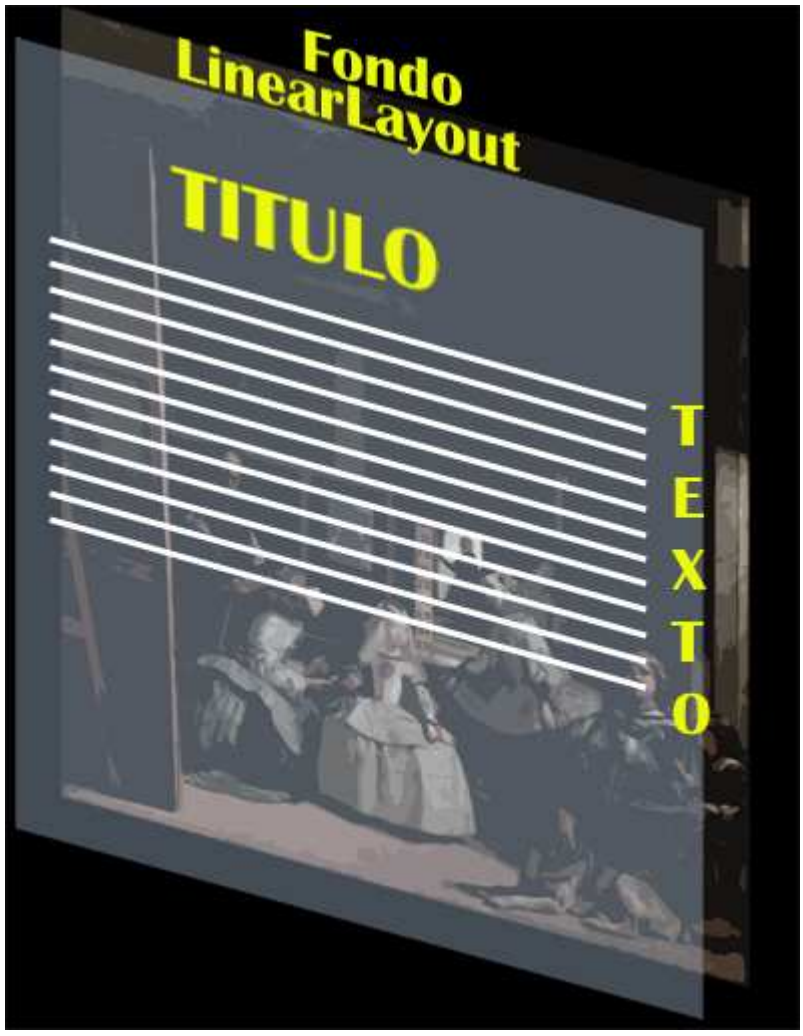


Fig.5.2.3.1.4 Esquema capas de info.xml

Una vez hemos implementado todo el código, realizamos pruebas directamente sobre esta Activity, ya que la Activity de la primer parte de la aplicación no está construida. En resumen, probamos el correcto funcionamiento de la navegación entre los elementos del cuadro, observando durante estos test que convendría añadir una serie de botones al final del texto que nos realizaran la misma función. Esto se debe a que para el correcto funcionamiento del método onTouch combinado con un texto *"scrollable"* (al deslizar el dedo de arriba a abajo o viceversa avancemos/retrocedamos en el texto) hemos de realizar el movimiento casi perfecto, ya que en otro caso el evento *"scrollable"* es el que ejerce su función.

Segunda Iteración

En nuestra segunda iteración buscaremos resolver los problemas surgidos en la anterior "vuelta" y añadiremos nuevos elementos y funcionalidades al proyecto. Para ello, comenzaremos pensando y diseñando una nueva forma de interactuar con los componentes reconocidos dentro del cuadro en cuestión de forma que nos permitan acceder la segunda parte de la aplicación pulsando sobre ellos con los datos necesarios para su identificación y presentación. Tras un tiempo barajando

posibilidades, convenimos que la mejor forma de realizarlo consistiría en mostrar un “cartel” con el nombre del componente encima del mismo y que este realice la función de botón que nos lance la *Activity*de información (info.java). Para ello:

Crearemos un nuevo fichero de texto en el que indicaremos el nombre del componente y las coordenadas, en porcentaje respecto al ancho y alto del cuadro, respecto al borde superior izquierdo del cuadro donde ha de posicionarse el “cartel” del componente en cuestión.

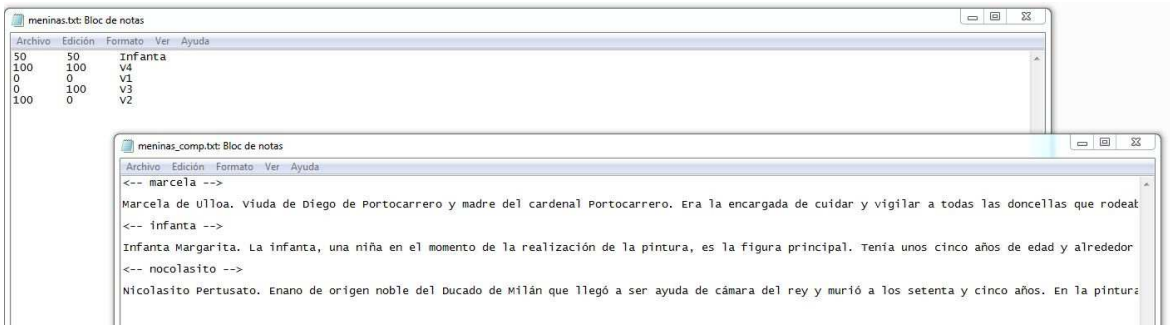


Fig. 5.2.3.2.1 Ficheros de Datos

Mediante una serie de métodos, pasaremos las coordenadas de la pantalla donde se sitúa la esquina superior izquierda del cuadro así como la anchura y altura del mismo respecto a la resolución de la pantalla.

- Cogemos las coordenadas del “*Trackable”* detectado. Lo procesamos y enviamos al módulo ImageTargetsRenderer.java:

```
//Sacar las coordenadas del trackable
const QCAR::ImageTarget* imageTarget =
                        static_cast<const QCAR::ImageTarget*> (trackable);
QCAR::Vec2F targetSize = imageTarget->getSize();
// Cache the projection matrix:
const QCAR::CameraCalibration& cameraCalibration =
                        QCAR::CameraDevice::getInstance().getCameraCalibration();
float halfWidth = targetSize.data[0] / 2.0f;
float halfHeight = targetSize.data[1] / 2.0f;

QCAR::Vec2F v1 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(-halfWidth, halfHeight, 0));
QCAR::Vec2F v2 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(halfWidth, halfHeight, 0));
QCAR::Vec2F v3 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(-halfWidth, -halfHeight, 0));
QCAR::Vec2F v4 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(halfWidth, -halfHeight, 0));

v1 = cameraPointToScreenPoint(v1);
v2 = cameraPointToScreenPoint(v2);
v3 = cameraPointToScreenPoint(v3);
v4 = cameraPointToScreenPoint(v4);

float angle1 = atan2((v2.data[1]-v1.data[1]),(v2.data[0]-v1.data[0]));
float angle2 = atan2((v4.data[1]-v3.data[1]),(v4.data[0]-v3.data[0]));
float width1 = sqrt(pow((v2.data[0]-v1.data[0]),2)+pow((v2.data[1]-v1.data[1]),2));
float width2 = sqrt(pow((v4.data[0]-v3.data[0]),2)+pow((v4.data[1]-v3.data[1]),2));
float param[8] = {v1.data[0],v1.data[1],v3.data[0],v3.data[1],width1,width2,angle1,angle2};

jclass javaClass = env->GetObjectClass(obj);
jfloatArray jf = env->NewFloatArray(8);
env->SetFloatArrayRegion(jf,0,8,param);
jmethodID method = env->GetMethodID(javaClass, "setParam", "([F)V");
env->CallVoidMethod(obj, method,jf);
```

Fig. 5.2.3.2.2 ImageTargets.cpp

- En este módulo creamos un método de acceso a los valores almacenados para que desde el módulo principal, ImageTargets.java, podamos leerlos.

```
static float[] coord = new float[8]; //Altura, anchura y angulo
//Guardado y retorno de coordenadas
public void setParam(float[] cord){
    for(int i=0; i<8; i++){
        coord[i] = cord[i];
    }
    Message message = new Message();
    message.obj = "new_cord";
    mainActivityHandler.sendMessage(message);
}
public static float[] getParam(){
    return coord;
}
```

Fig. 5.2.3.2.3 ImageTargetsRenderer.java

- Ya podemos acceder desde el método principal a los datos necesarios para realizar la construcción de la estructura necesaria para la representación del cartel en la posición correcta.

```
protected int[] coordenadas(float cX, float cY){
    int[] cord = {0,0};
    float[] param = ImageTargetsRenderer.getParam();  //v1.data[0],v1.data[1],v3.data[0],v3.data[1],width1,width2,angle1,angle2
    double[] p1={0,0},p2={0,0};
    DebugLog.LOGD("Angulo: "+param[6]+","+param[7]);
    //Calculamos la posicion respecto al borde superior.
    p1[0] = param[0] + param[4] * cX/100.0 * FloatMath.cos(param[6]);
    p1[1] = param[1] + param[4] * cX/100.0 * FloatMath.sin(param[6]);
    //Calculamos la posicion respecto al borde inferior.
    p2[0] = param[2] + param[5] * cX/100.0 * FloatMath.cos(param[7]);
    p2[1] = param[3] + param[5] * cX/100.0 * FloatMath.sin(param[7]);

    //Sacamos la coordenadas finales del punto uniendo los 2 puntos.
    double angle =  Math.atan2((p2[0]-p1[0]) , (p2[1]-p1[1])); //Sacamos el angulo que los une.
    double height = Math.sqrt( Math.pow(p2[1]-p1[1],2) + Math.pow(p2[0]-p1[0],2)); //Sacamos la distancia

    //Sacamos finalmente las coordenadas.
    cord[0] = (int)(p1[0] + (height * cY/100.0 * Math.sin(angle)));
    cord[1] = (int)(p1[1] + (height * cY/100.0 * Math.cos(angle)));

    return cord;
}
```

Fig. 5.2.3.2.4 ImageTargets.java

- Con ambos datos, las coordenadas dentro de la pantalla de la esquina del cuadro y las coordenadas respecto a esta del cartel, ya podemos situarlo en la posición correcta sobre el componente. Para ello realizamos una serie de operaciones matemáticas, las cuales observamos en la imagen anterior. Además, para conseguir posicionar los “carteles” habremos de crear una estructura de “Layouts”:
    - Un *“RelativeLayout”* que a raíz del padre, el *“LinearLayout”* de fondo, nos permita situar cualquier elemento hijo a partir de la esquina superior izquierda y no siguiendo el esquema de su padre. De esta forma, le

proporcionamos unos márgenes que nos hagan de coordenadas para el posicionamiento de los hijos del Layout.

- Introduciremos el nombre del componente o elemento del cuadro en un *"TextView"* . Asignaremos como atributo "*Tag"* del mismo el propio nombre del objeto en cuestión y el del cuadro al que pertenece para conservarlo en la llamada a la *Intent*que nos ejecuta la *Activity*de información. De tal forma que nos quedaría en pantalla, de forma esquemática, algo así como la siguiente imagen:

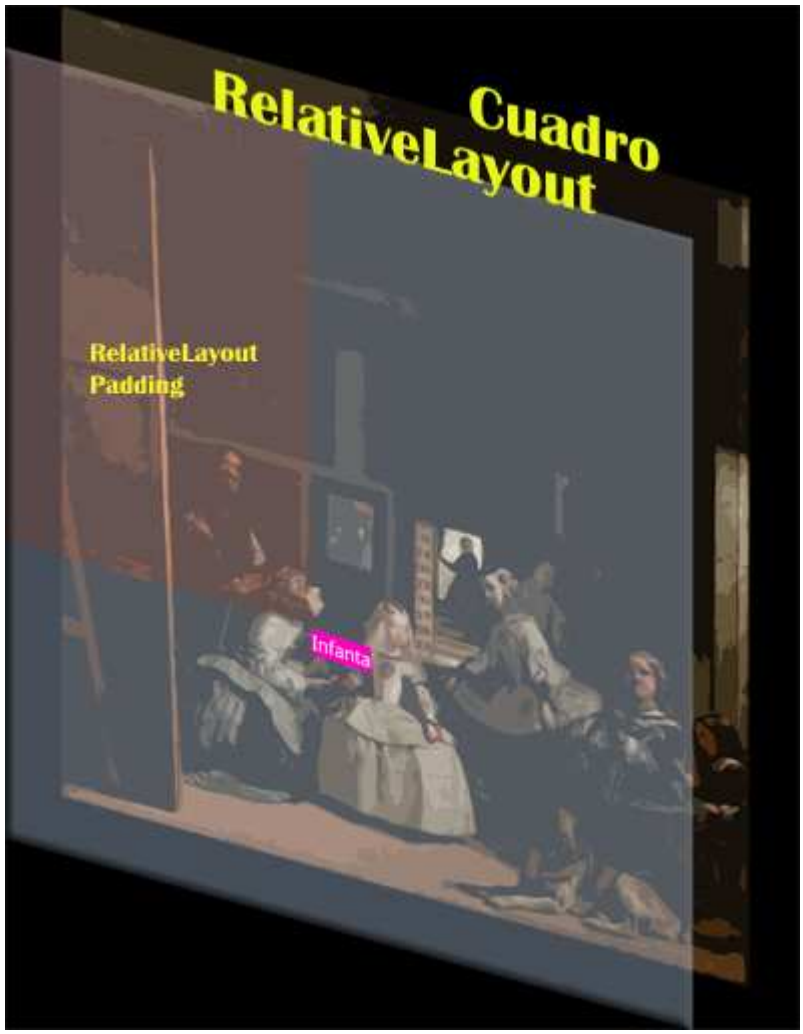


Fig. 5.2.3.2.5 Esquema capas APicture.java, visión por la cámara del cuadro.

- Además de todo ello, configuramos una serie de parámetros de los anteriores componentes de la interfaz gráfica para que el "RelativeLayout" sea invisible o transparente y podamos leer el cartel sin dificultad. Una vez hecho lo anterior, le añadimos la característica de ser pulsable y que al ser pulsado el cartel, se comporte como un botón lanzando la Activity "info.java" mediante un Intent*:*

```
//Abrimos el fichero
try {
    String text1 = "";
    int i = 0, cX, cY, color = Color.RED;

    String query = "SELECT cordX, cordY, Nombre FROM Carteles WHERE cuadro='"+text+"'";
    c = baseDatos.rawQuery(query,null);
    c.moveToFirst();
    // read every line of the file into the lines-variable, on line at the time
    while (!c.isAfterLast()) {
        // do something with the settings from the file
        // Legacy code:
         cX = c.getInt(c.getColumnIndex("cordX"));
         cY = c.getInt(c.getColumnIndex("cordY"));
         cord = coordenadas(cX,cY);
         text1= c.getString(c.getColumnIndex("Nombre"));
         rel[i] = (RelativeLayout) new RelativeLayout(context);
         rel[i].setId(i+1);
         carteles[i][0] = rel[i].getId();
         carteles[i][1] = cX;
         carteles[i][2] = cY;
         rel[i].setPadding(cord[0],cord[1],0,0);
         r1.addView(rel[i]);

         texto[i] = (TextView) new TextView(context);
         texto[i].setText(text1.trim());
         texto[i].setTag(text+"-"+text1.trim());
         texto[i].setTextColor(Color.WHITE);
         texto[i].setBackgroundColor(color);
         texto[i].setGravity(Gravity.BOTTOM);
         texto[i].setClickable(true);
         texto[i].setOnClickListener(new View.OnClickListener() {
              public void onClick(View v) {
                  Intent intent = new Intent(context, info.class);
                  StringTokenizer st = new StringTokenizer((String) v.getTag(), "-",false);
                  intent.putExtra("cuadro", st.nextToken());
                  intent.putExtra("componente", st.nextToken());
                  r1.removeAllViews();
                  startActivity(intent);
              }
          });
         color += i*75;

         rel[i].addView(texto[i]);
         i++;
         c.moveToNext();
    }
addContentView(r1,new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
```

Fig. 5.2.3.2.6 Código de Representación de los Carteles.

- En lo que respecta a la segunda parte del proyecto, hemos investigado cómo realizar la transición entre los componentes del fichero de una forma más visual y amigable, dando la sensación que pasas de diapositiva o página. Para ello, podemos ver en la Fig. como tenemos una línea con *"ly.startAnimation();"* al comienzo de ambas opciones que nos ejecuta una animación prediseñada. Como ya comentamos al final de la primer iteración, vamos a añadir un par de botones para realizar la navegación y otro para volver a la primera parte de la aplicación de forma controlada. Para esto último, hemos de seguir una serie de pasos que eviten errores en futuros accesos a la segunda parte, tales como cerrar el fichero de lectura, cerrar el cursor de lectura... etc.

```
public void onPreviousClick(View buttom){
    Animation animL = AnimationUtils.loadAnimation(this, R.anim.slide_left);
    //Movimiento de izq a drch
    ly.startAnimation(animL);
    try{
        //AVANZAR AL SIGUIENTE ELEMENTO

        //Ponemos el titulo
        if(!titulo.equals("none"))
            title.setText(titulo);
        else
            title.setText(cuadro);

        tv.setText(texto);
    }catch(Exception ex){
        DebugLog.LOGD("Excepcion1: "+ex.toString());
        title.setText("ERROR1");
        tv.setText("ERROR1");
    }
}
public void onNextClick(View buttom){
    Animation animR = AnimationUtils.loadAnimation(this, R.anim.slide_right);
    //Movimiento de drch a izq
    ly.startAnimation(animR);
    try{
        //RETROCEDER AL ANTERIOR ELEMENTO

        //Ponemos el titulo
        if(!titulo.equals("none"))
            title.setText(titulo);
        else
            title.setText(cuadro);

        tv.setText(texto);
    }catch(Exception ex){
        DebugLog.LOGD("Excepcion2: "+ex.toString());
        title.setText("ERROR2");
        tv.setText("ERROR2");
    }
}
```

```
public boolean onTouch(View v, MotionEvent event){
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            oldX = (int) event.getX();
            oldY = (int) event.getY();
            break;
        case MotionEvent.ACTION_UP:
            if ( (oldX > (int) event.getX()) ){
                onPreviousClick(null);
            }else if((oldX < (int) event.getX()) ){
                onNextClick(null);
            }
            break;
    }
    return true;
}
```

Fig. 5.2.3.2.7 Código botones info.java

Añadiendo al fichero XML el código:

```
<Button
    android:id="@+id/previous"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onPreviousClick"
    android:text="&lt;&lt;" />

<Button
    android:id="@+id/volver"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onVolverClick"
    android:text="Volver" />

<Button
    android:id="@+id/next"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onNextClick"
    android:text=">>" />
```

Fig. 5.2.3.2.8 Código botones XML

Fig. 5.2.3.2.9 Esquema info.xml

La codificación del nuevo diseño resultó bastante compleja debido a la necesidad de añadir a la visión de la cámara elementos de la interfaz gráfica de Android (RelativeLayout,TextView...etc). Durante el aprendizaje del lenguaje se trabajamos sobre nuestras propias Activity's y por lo tanto podíamos trabajar con todos esos elementos de manera personalizada y cómoda, pero al tener que adaptarnos al modelo de Vuforia, nos encontramos que muchos de los métodos seguidos para la representación de nuestros componentes no resultaron y tuvimos que recurrir a fuentes externas para la resolución de los problemas. Otro problema que nos encontramos fue que los TextView que utilizamos para mostrar los carteles resultan demasiado pequeños y complican la pulsación sobre el elemento en cuestión. Una posible solución podría ser aumentar tanto su tamaño como la fuente del texto, pero esto resultaría en unos carteles que taparían el propio objeto que señalan, por ello en la siguiente iteración buscamos una solución alternativa. Por último, durante las pruebas de la versión actual, nos damos cuenta de la inexactitud del método matemático de posicionamiento del cartel al no tener en cuenta la profundidad o eje-z a la que se sitúa el cuadro que nos provoca un posicionamiento incorrecto de los carteles. En la siguiente iteración se investiga su resolución.

En lo relativo a la pantalla de información, durante el desarrollo, hicimos varias versiones en cuanto a sobre qué zona habremos de deslizar el dedo para que naveguemos de un componente a otro del cuadro pero finalmente decidimos implementarlo sobre el propio texto pese a los problemas que habíamos detectado en la anterior iteración. Esta decisión la tomamos partiendo del hecho que la mayoría de la pantalla es ocupada por el texto y es sobre la que el usuario tenderá a realizar la acción. Además, observamos que la fuente elegida para el texto resulta ilegible en determinadas zonas al coincidir con el fondo. Para ello probamos diferentes fuentes y métodos para asignar de forma dinámica el color de la fuente, pero debido a su complejidad se opta por una fuente estática y un color lo más contrastado posible.

Una vez completado todos los objetos de la iteración y analizado el resultado de la misma, podemos plantearnos una nueva iteración, que en el caso será la definitiva.

Tercera Iteración

Durante esta iteración, la última que realizamos en el proyecto, se pretende llegar a una versión de la aplicación en la que tendremos, ahora sí, los datos almacenados en una BBDD así como un nuevo sistema de interacción con la aplicación en la primera parte del programa. Para ello, comenzaremos creando e implementando las bases de datos en la que almacenamos toda la información necesaria para el funcionamiento de la aplicación y que hasta ahora teníamos guardado en unos ficheros de texto. Posteriormente, una vez comprobado el correcto funcionamiento de la versión de la anterior iteración con BBDD, pasaremos al diseño del nuevo método de interacción entre el usuario y la aplicación en la primera parte.

Respecto a lo primero, pensamos que la mejor manera de trabajar con las BBDD desde nuestra aplicación será disponer de:

- Tabla Carteles

Se trata de una tabla en nuestra base de datos con la información acerca de cada uno de los componentes de cada cuadro que ha de disponer un “cartel” junto a las coordenadas de los 4 vértices del cuadrilátero que lo cubre (más adelante se explica).

X1	Y1	X2	Y2	X3	Y3	X4	Y4	Nombre_componente	Cuadro

- Tablas de cada Cuadro

Disponemos de una serie de tablas, una distinta para cada cuadro de nuestra aplicación, con toda aquella información que necesitamos en la segunda parte de la aplicación. Es decir, contiene tanto el nombre de los componentes del cuadro que podemos consultar como toda aquella información que consideremos interesante mostrar en pantalla. Además, tendremos una línea adicional con información del propio cuadro.

Nombre_componente	Información

De este modo, en cada ocasión solo cargaremos la información estrictamente necesaria, tanto en cada una de las 2 partes como en cuando vayamos a navegar entre los componentes de uno de los cuadros.

Al respecto del sistema de interacción con la aplicación en la primera parte, cuando pulsamos sobre la pantalla de nuestro terminal para obtener información sobre el componente del cuadro, hemos cambiado por completo su funcionamiento, hemos abandonado la idea de mostrar cuadros de texto con el nombre que corresponda y optamos por construir un cuadrilátero a partir de 4 puntos que abarque el componente de la mejor manera que seamos capaces. Así, partiremos de 4 coordenadas indicadas en la BBDD para, mediante lo que en programación Android se conoce como Canvas, calcular si las coordenadas sobre las que se detecta la pulsación está contenida entre los puntos y de esta forma lanzar la segunda parte con el componente correspondiente. Además de todo esto, reproduciremos el propio cuadro detectado en blanco y negro exceptuando las zonas pulsables, que irán bordadas en blanco. Para esto último, tenemos que retomar la representación de objetos 3D, pero esta vez no construiremos un objeto complejo, simplemente hacemos un plano que cubra el cuadro y la aplicamos la textura que corresponda (la del propio cuadro con zonas resaltadas). El esquema de los polígonos sobre la pantalla en blanco y negro quedaría de la siguiente forma

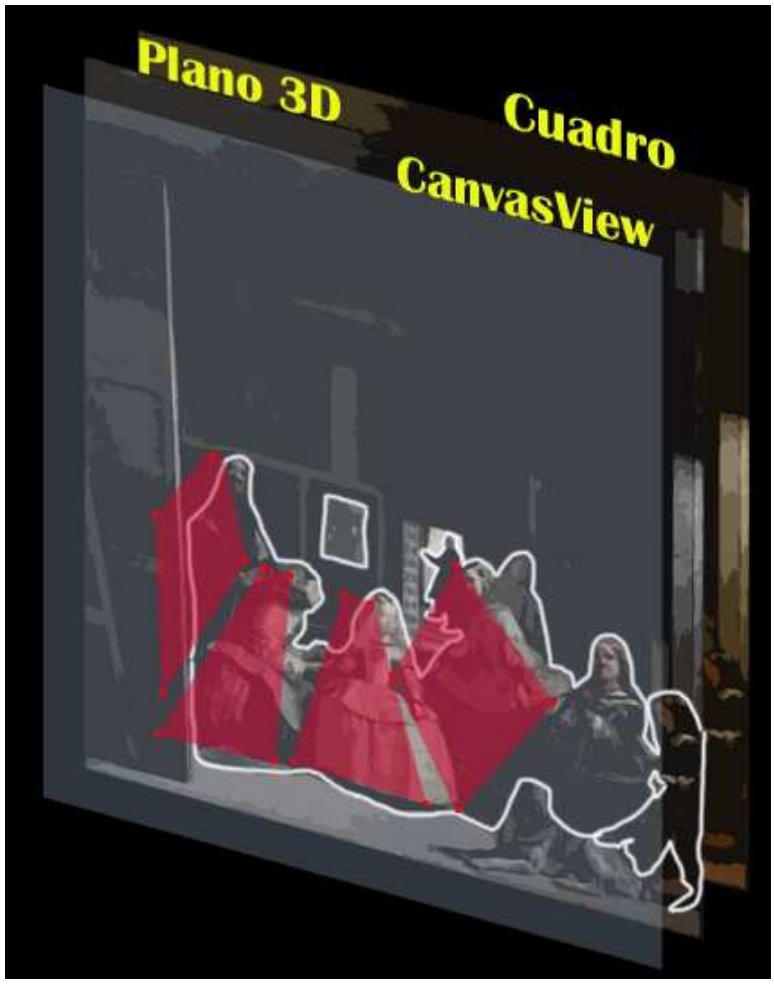


Fig. 5.2.3.3.1 Cuadriláteros pulsables.

En cuanto al problema que teníamos en la anterior iteración con el posicionamiento de las coordenadas de los carteles, al no ser capaces de hallar la profundidad a la que nos encontramos, decidimos utilizar el propio método de código nativo que nos daba las coordenadas de las 4 esquinas y la anchura del cuadro para el cálculo de esas coordenadas. De este modo haremos una consulta al método con cada conjunto de 4 vértices, es decir, con cada cartel. Esos valores los guardamos en variables de la clase para poder calcular posteriormente si las coordenadas sobre las que hemos pulsado están contenidas en el cuadrilátero formado por los vértices.

En la segunda parte de la aplicación, además del tema del uso de BBDD, vamos a introducir una imagen del componente en cuestión como fondo del texto de manera que sepamos a cuál de ellos nos estamos refiriendo.

Al terminar toda la implementación y comprobado su funcionamiento realizaremos la tarea de renombrar el proyecto con el nombre de nuestra aplicación y adoptaremos todas las referencias entre módulos. Hasta este momento no se había trabajado más que con el ejemplo proporcionado por Vuforia, modificándolo ahí donde se ha señalado que era necesario, se consideró mejor esperar a conocer con más profundidad el sistema para realizar este paso.

Para la implementación del nuevo diseño habremos de crear nuevas clases o módulos que nos realicen las funciones anteriormente descritas, así como modificar las presentes para que la arquitectura del sistema quede como se ha descrito en el apartado de diseño.

Finalmente, tendremos que implementar el siguiente código:

- ImageTargets.java (que pasa a llamarse APicture.java)

Hemos descargado prácticamente por completo el peso de este módulo respecto a las funcionalidades añadidas, todo aquello que le añadimos al ejemplo proporcionado por Vuforia. Ahora realiza la carga de la BBDD, llama a la creación de los carteles y por último muestra en un mensaje momentáneo de que cuadro se trata.

  - Cargamos las clases nuevas.

```
//DDBB
static SQLiteDatabase baseDatos = null;

//CanvasView:clase que nos crea las views del trackable
private CanvasView carteles;
```

Fig. 5.2.3.3.2 Carga de las clases nuevas.

  - En el método *onCreate()* intentamos crear la BBDD, si está creada (no es la primera vez que ejecutamos la app) sacamos un mensaje por el log

notificando que no se puede crear puesto que ya existe y si podemos hacemos lo mismo notificando que se ha creado la BBDD.

```
/** Called when the activity first starts or the user navigates back
 * to an activity. */
protected void onCreate(Bundle savedInstanceState)
{
    /*
    .

    //Create the DDBB
    DebugLog.LOGD("BBDD...");
    try{
        baseDatos = SQLiteDatabase.openDatabase("/data/data/com.qualcomm.QCARSamples.APicture/databases/MiProyecto.db" , null, SQLiteDatabase.OPEN_READONLY);
        DebugLog.LOGD("Existe");
    }catch(SQLException ex){
        DebugLog.LOGD("No Existe:"+ex.toString());
        new BBDD(getApplicationContext());
        // create or open database file
        baseDatos = BBDD.getBBDD();
    }finally{
        baseDatos.close();
    }

    // Update the application status to start initializing application
    updateApplicationStatus(APPSTATUS_INIT_APP);
}
```

Fig. 5.2.3.3.3 onCreate()

- En el método onResume() captamos el mensaje que nos llega, leemos de que cuadro se trata, llamamos al constructor del *CanvasView()*, añadimos el resultado a la escena en pantalla y finalmente mostramos un cartel con el nombre del cuadro que hemos detectado. La construcción del *CanvasView()* resulta invisible puesto que no nos interesa que “tape” la representación del cuadro con zonas resaltadas.

```
// Create a new handler for the renderer thread to use
// This is necessary as only the main thread can make changes to the UI
ImageTargetsRenderer.mainActivityHandler = new Handler() {
    final Context context = getApplicationContext();

    @Override
    public synchronized void handleMessage(Message msg) {
        String text = (String) msg.obj;
        int duration = Toast.LENGTH_SHORT;
        try {
            carteles = new CanvasView(context,text);
            addContentView(carteles,new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
            Toast toast = Toast.makeText(context, text, duration);
            toast.show();
        } catch (Exception e) {
            // do something if the myfilename.txt does not exits
            System.out.println("Excepcion->"+e.toString());

            Toast toast = Toast.makeText(context, text, duration);
            toast.show();
        }
    }
};
```

Fig. 5.2.3.3.4 Handler()

- ImageTargets.cpp (que pasa a llamarse APicture.cpp)

En este módulo hemos es donde hemos de retomar la idea de realizar una representación 3D del cuadro sobre un plano. Para ello, definimos una series de parámetros y valores que necesitar OpenGL para su construcción; que son las

coordenadas de los 4 vértices, de las texturas, las normales del plano y por último los índices.

```
static const float planeVertices[] = { -0.5, -0.5, 0.0, 0.5, -0.5, 0.0, 0.5, 0.5, 0.0, -0.5, 0.5, 0.0, };

static const float planeTexcoords[] = { 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0 };

static const float planeNormals[] = { 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0 };

static const unsigned short planeIndices[] = { 0, 1, 2, 0, 2, 3 };
```

Fig. 5.2.3.3.5 Valores para la creación del plano 3D

Y sustituimos el código de renderizado de los objetos 3D presente hasta el momento en este fichero, por el siguiente:

```
// assuming this is an image target
const QCAR::ImageTarget* imageTarget =
                        static_cast<const QCAR::ImageTarget*> (trackable);
QCAR::Vec2F targetSize = imageTarget->getSize();

QCAR::Matrix44F modelViewProjection;

SampleUtils::translatePoseMatrix(0.0f, 0.0f, kObjectScale, &modelViewMatrix.data[0]);

SampleUtils::scalePoseMatrix(targetSize.data[0], targetSize.data[1], 1.0f, &modelViewMatrix.data[0]);
SampleUtils::multiplyMatrix(&projectionMatrix.data[0], &modelViewMatrix.data[0] , &modelViewProjection.data[0]);

glUseProgram(shaderProgramID);
glVertexAttribPointer(vertexHandle, 3, GL_FLOAT, GL_FALSE, 0, (const GLvoid*) &planeVertices[0]);
glVertexAttribPointer(normalHandle, 3, GL_FLOAT, GL_FALSE, 0, (const GLvoid*) &planeNormals[0]);

glVertexAttribPointer(textureCoordHandle, 2, GL_FLOAT, GL_FALSE, 0, (const GLvoid*) &planeTexcoords[0]);

glEnableVertexAttribArray(vertexHandle);

glEnableVertexAttribArray(normalHandle);

glEnableVertexAttribArray(textureCoordHandle);

glActiveTexture(GL_TEXTURE0);

glBindTexture(GL_TEXTURE_2D, thisTexture->mTextureID);
glUniformMatrix4fv(mvpMatrixHandle, 1, GL_FALSE, (GLfloat*)&modelViewProjection.data[0] );
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_SHORT, (const GLvoid*) &planeIndices[0]);
```

Fig. 5.2.3.3.6 Creación del plano 3D

Luego, modificamos el método que teníamos para hallar las coordenadas de las esquinas, la altura y la anchura, por un método que nos transforme las coordenadas respecto al plano en coordenadas respecto a la pantalla, de forma que podamos llamarlo con cualquier valor en porcentaje respecto al ancho y alto del cuadro y nos diga qué coordenadas son en la pantalla de nuestro terminal.

```
JNIEXPORT jintArray JNICALL
Java_com_qualcomm_QCARSamples_APicture_Cartel_getCoordinates(JNIEnv* env,jobject obj, jdoubleArray coord)
{
    //Variables para pasar de Vec2F a int
    int c1[2],c2[2],c3[2],c4[2],p2[8];
    //variable Resultado a devolver
    jintArray result;

    result = env->NewIntArray(8);

    // Get the state from QCAR and mark the beginning of a rendering section
    QCAR::State state = QCAR::Renderer::getInstance().begin();
        // Did we find any trackables this frame?
    for(int tIdx = 0; tIdx < state.getNumActiveTrackables(); tIdx++)
    {
        // Get the trackable:
        const QCAR::Trackable* trackable = state.getActiveTrackable(tIdx);

        jdouble *body = env->GetDoubleArrayElements(coord,0);

        //Sacar las coordenadas del trackable
        const QCAR::ImageTarget* imageTarget =
                                    static_cast<const QCAR::ImageTarget*> (trackable);
        QCAR::Vec2F targetSize = imageTarget->getSize();

        // Cache the projection matrix:
        const QCAR::CameraCalibration& cameraCalibration =
                                    QCAR::CameraDevice::getInstance().getCameraCalibration();

        for(int j=0; j<4; j++){
            if(body[j*2]<50){
                p2[j*2] = -(targetSize.data[0] * ((float)(50-body[j*2])/100.0f));
            }else{
                p2[j*2] = targetSize.data[0] * ((float)(body[j*2]-50)/100.0f);
            }
            if(body[j*2+1]<50){
                p2[j*2+1] = targetSize.data[1] *((float)(50-body[j*2+1])/100.0f);
            }else{
                p2[j*2+1] = -(targetSize.data[1] *((float)(body[j*2+1]-50)/100.0f));
            }
        }

        QCAR::Vec2F v1 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(p2[0], p2[1], 0));
        QCAR::Vec2F v2 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(p2[2], p2[3], 0));
        QCAR::Vec2F v3 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(p2[4], p2[5], 0));
        QCAR::Vec2F v4 = QCAR::Tool::projectPoint(cameraCalibration, trackable->getPose(), QCAR::Vec3F(p2[6], p2[7], 0));

        v1 = cameraPointToScreenPoint(v1);
        c1[0] = v1.data[0];
         c1[1] = v1.data[1];

         v2 = cameraPointToScreenPoint(v2);
         c2[0] = v2.data[0];
         c2[1] = v2.data[1];

         v3 = cameraPointToScreenPoint(v3);
         c3[0] = v3.data[0];
         c3[1] = v3.data[1];

         v4 = cameraPointToScreenPoint(v4);
         c4[0] = v4.data[0];
         c4[1] = v4.data[1],

         env->ReleaseDoubleArrayElements(coord, body, 0);
         env->SetIntArrayRegion(result, 0, 2, c1);
         env->SetIntArrayRegion(result, 2, 2, c2);
         env->SetIntArrayRegion(result, 4, 2, c3);
         env->SetIntArrayRegion(result, 6, 2, c4);
    }

    return result;
}
```

Fig. 5.2.3.3.7 getCoordinates()

- ImageTargetsRenderer.java

  Puesto que la consulta de las coordenadas en pantalla se realiza ahora desde otra clase, hemos borrado los dos métodos que teníamos en este módulo que

realizaban la función de intermediario entre las dos clases anteriores. Por lo tanto, este módulo sólo añadirá los métodos de paso de mensajes entre las clases Java y C++ al código original obtenido del SDK Vuforia.

- BBDD.java

  Se trata, como se puede deducir del nombre, el módulo que se encarga de crear y poblar las BBDD. Como se ha comentado antes, será llamado al comienzo de la ejecución de la aplicación si y sólo sí es la primera vez que se ejecuta la misma, de tal forma que si ya existe no se intenten crear o machacar la base de datos. Por ello mismo, todo cambio en la misma nos supone tener que reinstalar toda la aplicación desde 0. El módulo se compone de un constructor de la clase dividido en 3 partes:

  - Construcción de la BBDD

    Creamos la base de datos con un nombre que permita identificarla posteriormente.

```
//DDBB
static SQLiteDatabase baseDatos = null;

public BBDD(Context contx){
    DebugLog.LOGD("Creando/Poblando tablas BBDD");
    try{
        // create or open database file
        baseDatos = contx.openOrCreateDatabase("MiProyecto.db" , SQLiteDatabase.CREATE_IF_NECESSARY, null);
        DebugLog.LOGD("Base de datos no existente.");
        baseDatos.setVersion(1);
        baseDatos.setLocale(Locale.getDefault());
        baseDatos.setLockingEnabled(true);
```

Fig. 5.2.3.3.8 Crear la BBDD

  - Creación de las tablas de cada cuadro

Creamos una tabla para cada cuadro que tendremos reconocible. En nuestro caso, hemos creado solo 2 tablas para cada uno de los cuadros de Velázquez que reconoce nuestra aplicación y poblamos ambas tablas.

```
//RENDICION DE BREDA
baseDatos.execSQL("CREATE TABLE Breda (Id INTEGER PRIMARY KEY AUTOINCREMENT, Nombre TEXT, Informacion TEXT)");

//Poblar las tablas
ContentValues values = new ContentValues();
values.put("Nombre", "Justino de Nassau");
values.put("Informacion", "Justino de Nassau (? 1559 - Leiden, 1631) fue hijo de Guillermo de Orange, estatúder de las Provincias Unidas de los Países Bajos y de su am
baseDatos.insert("Breda", null, values);

^
~~^
.
//MENINAS
baseDatos.execSQL("CREATE TABLE Meninas (Id INTEGER PRIMARY KEY AUTOINCREMENT, Nombre TEXT, Informacion TEXT)");

//Poblar las tablas
values = new ContentValues();
values.put("Nombre", "Infanta");
values.put("Informacion", "Infanta Margarita. La infanta, una niña en el momento de la realización de la pintura, es la figura principal. Tenía unos cinco años de edad
baseDatos.insert("Meninas", null, values);

^
~~^
.
```

Fig. 5.2.3.3.9 Crear y poblar las tablas de cada cuadro

- Creación de la tabla de carteles

En esta tabla tendremos las coordenadas de los 4 vértices de cada “cartel” o cuadrilátero que hacen de botón en la primer parte y el nombre del componente.

```
baseDatos.execSQL("CREATE TABLE Carteles (cx1 REAL,cy1 REAL, cx2 REAL,cy2 REAL, cx3 REAL,cy3 REAL, cx4 REAL,cy4 REAL,Nombre TEXT, Cuadro TEXT)");

//Poblar las tablas
ContentValues values2 = new ContentValues();

//Justino de Nassau
values2.put("cx1", "45.31");
values2.put("cy1", "45.45");
values2.put("cx2", "50.31");
values2.put("cy2", "46.21");
values2.put("cx3", "49.06");
values2.put("cy3", "89.54");
values2.put("cx4", "28.75");
values2.put("cy4", "86.36");
values2.put("Nombre", "Justino de Nassau");
values2.put("Cuadro", "Breda");
baseDatos.insert("Carteles", null, values2);


//Poblar las tablas
values2 = new ContentValues();

//Maribarbola
values2.put("cx1", "82.14");
values2.put("cy1", "61.18");
values2.put("cx2", "86.57");
values2.put("cy2", "61.61");
values2.put("cx3", "92.85");
values2.put("cy3", "93.16");
values2.put("cx4", "74.28");
values2.put("cy4", "84.47");
values2.put("Nombre", "Maribarbola");
values2.put("Cuadro", "Meninas");
baseDatos.insert("Carteles", null, values2);
```

Fig. 5.2.3.3.10 Crear y poblar la tabla Carteles

Por último, disponemos de un método que nos devuelve el identificador de la base de datos para que podamos acceder a la misma desde los módulos que lo necesiten.

```
public static SQLiteDatabase getBBDD(){
    return baseDatos;
}
```

Fig. 5.2.3.3.11 getBBDD()

- CanvasView.java

Es la clase instanciada por *ImageTargets.java* cuando recibe el mensaje de que ha detectado uno de los cuadros. En ella, construimos lo que podría entenderse como una “pizarra” transparente sobre la que dibujos todo lo que queramos, en nuestro caso serán los cuadriláteros de los carteles. Para ello, accedemos a la base de datos y recorremos todas las filas llamando, con cada una, a la clase *Cartel.java* y al terminar con todas llamamos con las coordenadas de los 4 vértices del cuadro.

```
public CanvasView(final Context context,final String pic) {
    super(context);
    cntx = context;
    picture = pic;
    DebugLog.LOGD("CanvasView");
    int i=0;
    Cursor c;

    setOnTouchListener(this);

    //Leemos de la BBDD
    SQLiteDatabase baseDatos = SQLiteDatabase.openDatabase("/data/data/com.qualcomm.QCARSamples.APicture/databases/MiProyecto.db" , null, SQLiteDatabase.OPEN_READONLY);
    String query = "SELECT cx1, cy1, cx2, cy2, cx3, cy3, cx4, cy4, Nombre FROM Carteles WHERE cuadro='"+pic+"'";
    c = baseDatos.rawQuery(query,null);
    c.moveToFirst();

    while (!c.isAfterLast()) {
        Cartel cart;
        //guardamos las coordenadas del cartel correspondiente
        double[][] coordenadas = {{c.getDouble(c.getColumnIndex("cx1")), c.getDouble(c.getColumnIndex("cy1"))},{c.getDouble(c.getColumnIndex("cx2")), c.getDouble(c.getColumnInd
        cart = new Cartel(c.getString(c.getColumnIndex("Nombre")),coordenadas);

        c.moveToNext();
        carteles[i++] = cart;
    }
    double[][] coordenadas = {{0.0,0.0},{100.0,0.0},{100.0,100.0},{0.0,100.0}};
    carteles[i++] = new Cartel("none",coordenadas);
    elementos = i;
    c.close();
    baseDatos.close();
}
```

Fig. 5.2.3.3.12 Constructor de CanvasView.java

Una vez tenemos todos los “carteles” creados, podemos dibujarlos mediante la llamada al método *onDraw()* que se realiza implícitamente en el *addContentView*de *ImageTargets.java* (APicture.java). Lo creamos de forma transparente para permitir que se vea la recreación del cuadro en blanco y negro sin tapar a los componentes. Por último, para que se actualice continuamente añadimos la orden *invalidate().*

```
@Override
public void onDraw(Canvas canvas) {
    Paint paint = new Paint();
    paint.setARGB(0,0,0,0);
    paint.setStyle(Paint.Style.FILL);
    canvas.drawColor(0, Mode.CLEAR);
    int j=0;
    while(j < elementos){
        canvas.drawPath(carteles[j].getPath(),paint);
        j++;
    }
    invalidate();
}
```

Fig. 5.2.3.3.13 onDraw()r de CanvasView.java

Además, esta es la clase que permanecerá atenta a si pulsamos la pantalla. Implementamos el *“setOnTouchListener(this)”* para ello y capturamos el evento con el método *“onTouch()”*. Al detectar que hemos pulsado, realizamos una pasada llamando a todos los carteles preguntando si el punto está dentro de cada uno de ellos y cuando detectamos que está dentro de alguno llamamos a la segunda parte de la aplicación con la información que corresponda.

```
public boolean onTouch(View view, MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            //DebugLog.LOGD("Click en: "+(int)event.getX()+","+(int)event.getY());
            int j=0;
            boolean dentro=false;

            Intent intent = new Intent(cntx, info.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            //DebugLog.LOGD("Elementos: "+elementos);
            while(j < elementos && !dentro){
                dentro = carteles[j].isInside(new Point((int)event.getX(),(int)event.getY()));
                j++;
            }
            if(dentro){
                intent.putExtra("cuadro", picture);
                intent.putExtra("componente", carteles[j-1].nombre);
            }
            cntx.startActivity(intent);
    }
    return true;
}
```

Fig. 5.2.3.3.14 onTouch() de CanvasView.java

- Cartel.java

Es en esta clase donde, con el nuevo diseño de funcionamiento de la aplicación, realizamos la consulta a *ImageTargets.cpp* (APicture.cpp) sobre las coordenadas de representación en pantalla de una posición dentro del cuadro, por lo que tendremos que realizar llamadas a código nativo y esto nos supone más código a implementar en el módulo.

```
// Name of the native dynamic libraries to load:
private static final String NATIVE_LIB_SAMPLE = "APicture";
public native int[] getCoordinates(double[] cord);
static
{
    com.qualcomm.QCARSamples.APicture.APicture.loadLibrary(NATIVE_LIB_SAMPLE);
}
```

Fig. 5.2.3.3.15 Llamada a código nativo

Al detectar uno de nuestros cuadros, esta clase es llamada tantas veces como componentes tengamos para que nos construya una instancia con cada uno. Para el constructor necesitaremos recibir tanto las coordenadas de los 4 vértices como el nombre del componente que representa.

```
public Cartel(String nombre, double[][] coordenadas) {

    DebugLog.LOGD("Cartel de "+nombre);
     this.nombre=nombre;
     parm[0] = coordenadas[0][0];
     parm[1] = coordenadas[0][1];
     parm[2] = coordenadas[1][0];
     parm[3] = coordenadas[1][1];
     parm[4] = coordenadas[2][0];
     parm[5] = coordenadas[2][1];
     parm[6] = coordenadas[3][0];
     parm[7] = coordenadas[3][1];
}
```

Fig. 5.2.3.3.16 getBBDD()

Tendremos un método que nos devolverá el *"path"* que nos permita dibujar el cuadrilátero a partir de los vértices del objeto.

```
public Path getPath() {
        //Calculamos los puntos continuamente
        int[] cord = getCoordinates(parm);
        v1.x=cord[0];
        v1.y=cord[1];
        v2.x=cord[2];
        v2.y=cord[3];
        v3.x=cord[4];
        v3.y=cord[5];
        v4.x=cord[6];
        v4.y=cord[7];

        Point point1_draw = new Point(v1.x,v1.y);
        Point point2_draw = new Point(v2.x,v2.y);
        Point point3_draw = new Point(v3.x,v3.y);
        Point point4_draw = new Point(v4.x,v4.y);

        path = new Path();
        path.setFillType(Path.FillType.EVEN_ODD);
        path.moveTo(point1_draw.x,point1_draw.y);
        path.lineTo(point2_draw.x,point2_draw.y);
        path.lineTo(point3_draw.x,point3_draw.y);
        path.lineTo(point4_draw.x,point4_draw.y);
        path.lineTo(point1_draw.x,point1_draw.y);
        path.close();

        return path;
}
```

Fig. 5.2.3.3.18 getPath()

Por último, cuando pulsemos la pantalla la clase *CanvasView.java* nos consultará si el punto está dentro del cuadrilátero. Para ello creamos el método siguiente que nos calcula mediante geometría si cumple las condiciones teniendo en cuenta, además, la posible rotación de la pantalla y el ángulo de visualización.

```
public boolean isInside(Point point){
    double m1,m2,m3,m4;
    double b1, b2, b3, b4;
    boolean recta1,recta2,recta3,recta4;

    //Calculamos las pendientes de los lados
   try{
       m1=(double)(v2.y-v1.y)/(double)(v2.x-v1.x);
   }catch (Exception ex){
       m1=(double)(v2.y-v1.y)/(double)(v2.x-v1.x+0.000000001);
   }
   try{
       m2=(double)(v3.y-v2.y)/(double)(v3.x-v2.x);
   }catch(Exception ex){
       m2=(double)(v3.y-v2.y)/(double)(v3.x-v2.x+0.000000001);
   }
   try{
       m3=(double)(v4.y-v3.y)/(double)(v4.x-v3.x);
   }catch (Exception ex){
       m3=(double)(v4.y-v3.y)/(double)(v4.x-v3.x+0.000000001);
   }
   try{
       m4=(double)(v1.y-v4.y)/(double)(v1.x-v4.x);
   }catch (Exception ex){
       m4=(double)(v1.y-v4.y)/(double)(v1.x-v4.x+0.000000001);
   }
  //Ecuaciones de la recta, hay que hayar las b's
  b1=v1.y-v1.x*m1;
  b2=v2.y-v2.x*m2;
  b3=v3.y-v3.x*m3;
  b4=v4.y-v4.x*m4;

 if(m1<0){
      recta1 =((point.y-point.x*m1) < b1);
      if((v2.y - v1.y)<0){
          recta1 = !recta1;
      }
  }else{
      recta1 = ((point.y-point.x*m1)>b1);
      if((v2.y - v1.y)<0){
          recta1 = !recta1;
      }
  }

  if(m2<0){
      recta2 =((point.y-point.x*m2) < b2);
      if((v3.y - v2.y)<0){
          recta2 = !recta2;
      }
  }else{
      recta2 = ((point.y-point.x*m2)>b2);
      if((v3.y - v2.y)<0){
          recta2 = !recta2;
      }
  }
```

```
        if(m3<0){
            recta3 =((point.y-point.x*m3) < b3);
            if((v4.y - v3.y)<0){
                recta3 = !recta3;
            }
        }else{
            recta3 = ((point.y-point.x*m3)>b3);
            if((v4.y - v3.y)<0){
                recta3 = !recta3;
            }
        }

        if(m4<0){
            recta4 =((point.y-point.x*m4) < b4);
            if((v1.y - v4.y)<0){
                recta4 = !recta4;
            }
        }else{
            recta4 = ((point.y-point.x*m4)>b4);
            if((v1.y - v4.y)<0){
                recta4 = !recta4;
            }
        }

        //Si cumple las 4 ecuaciones de la recta esta dent
        if(recta1 && recta2 && recta3 && recta4){
            return true;
        }else{
            return false;
        }
    }
```

Fig. 5.2.3.3.17 isInside()

- Info.java

En esta clase primero hemos sustituido el uso del fichero por el uso de las BBDD. El cambio ha supuesto tener que utilizar cursores para navegar entre los componentes y conservar donde nos quedamos para que cuando realizamos, si lo hacemos, un giro de la pantalla, no se reinicie desde el componente con el que entramos.

```
public void onVolverClick(View buttom){
    c.close();
    baseDatos.close();
    finish();
}
```

Fig. 5.2.3.3.19 Nuevo onVolverClick()

```
public void onPreviousClick(View buttom){
    Animation animL = AnimationUtils.loadAnimation(this, R.anim.slide_left);
    //Movimiento de izq a drch
    ly.startAnimation(animL);
    try{
        if(!c.moveToNext()){
            c.moveToFirst();
        }
        String titulo = c.getString(c.getColumnIndex("Nombre"));
        String texto = c.getString(c.getColumnIndex("Informacion"));

        .
        .
        .
}
public void onNextClick(View buttom){
    Animation animR = AnimationUtils.loadAnimation(this, R.anim.slide_right);
    //Movimiento de drch a izq
    ly.startAnimation(animR);
    try{
        if(!c.moveToPrevious()){
            c.moveToLast();
        }

        String titulo = c.getString(c.getColumnIndex("Nombre"));
        String texto = c.getString(c.getColumnIndex("Informacion"));
        .
        .
        .
```

Fig. 5.2.3.3.20 Nuevos métodos de navegación.

```
public void onCreate(Bundle savedInstanceState){
    Animation animacion = AnimationUtils.loadAnimation(this, R.anim.slide_left);
    // create or open database file
    baseDatos = openOrCreateDatabase("MiProyecto.db" , SQLiteDatabase.OPEN_READWRITE, null);
    ^
^^^^^^^
^^^^^^^
    try{
        String query = "SELECT Id, Nombre, Informacion FROM '"+cuadro+"'";
        c = baseDatos.rawQuery(query,null);
        c.moveToFirst();

        while(!c.getString(c.getColumnIndex("Nombre")).toLowerCase().replace(" ","").equals(componente.toLowerCase().replace(" ","")) ){
            c.moveToNext();
        }
        id = c.getInt(c.getColumnIndex("Id"));
        String texto = c.getString(c.getColumnIndex("Informacion"));
        .
        .
        .
}
```

Fig. 5.2.3.3.21 Nuevo onCreate()

Hecho el tránsito entre ficheros y BBDD, vamos a implementar el código para que nos aparezca la imagen del componente en el que estamos como fondo del texto.

- Primero hemos de modificar el fichero XML para incluir un ImageView dentro del scrollable y como fondo del texto, para lo que a su vez tendremos que meterlo todo en un RelativeLayout.

```
<ScrollView
    android:visibility="visible"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:keepScreenOn="true"
    android:scrollbarStyle="insideInset"
    android:layout_height="wrap_content" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <ImageView
            android:id="@+id/fondo_texto"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:adjustViewBounds="true"
            android:scaleType="fitCenter" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:isScrollContainer="true"
            android:orientation="vertical"
            android:scrollbars="vertical" >

            <TextView
                android:id="@+id/texto"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:paddingBottom="25sp"
                android:textColor="@android:color/white"
                android:textSize="18sp" />
```

Fig. 5.2.3.3.22 Inserción de la imagen en inf.xml

Después, tenemos que añadir el código en el fichero para que nos asigne qué imagen es la correcta.

```
//Asignamos la imagen correspondiente
int img = getResources().getIdentifier(cuadro.toLowerCase()+"_"+componente.toLowerCase().replace
DebugLog.LOGD("Imagen: "+cuadro.toLowerCase()+"_"+componente.toLowerCase().replace(" ",""));
fnd_texto.setImageResource(img);
```

Fig. 5.2.3.3.23 Asignación de la imagen.

Además, hemos sobrescrito la función de uno de los botones del terminal, para que al pulsarlo se ejecute nuestro método de finalización de la Activity. Esto lo hicimos para evitar problemas a la hora de volver a entrar a la segunda parte de la Activity tras pasar de nuevo por la primera.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    switch(keyCode)
    {
    case KeyEvent.KEYCODE_BACK:
          onVolverClick(null);
          return true;
    }
    return false;
}
```

Fig. 5.2.3.3.24 Método de sobrescritura del botón “Back”

- Renombrado del Proyecto

Una vez realizado todos los apartados anteriores, disponemos ya de la versión funcional final de nuestro proyecto, la aplicación que perseguimos con el desarrollo del proyecto está ya prácticamente acabada, solo nos queda realizar una serie de cambios con el objeto de darle un nombre propio al mismo y asignar los iconos de ejecución y carga de la aplicación. Con este fin realizaremos:

o Cambio del nombre de la aplicación.

Lo primero que hacemos para cambiar el nombre de la aplicación y conservar el correcto funcionamiento de todo el sistema es sustituir en el *AndroidManifest* el nombre del ejemplo utilizado, *ImageTargets,* por el propio, *APicture,* así como en el fichero *res/values/strings.xml.*

```
<activity android:label="@string/app_name"
          android:configChanges="orientation|keyboardHidden"
          android:name=".APicture">
```

Fig. 5.2.3.3.25 Cambio del nombre de la aplicación en *AndroidManifest*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">APicture</string>
    <string name="info">Informacion</string>
    <string name="app_description">QCAR APicture Sample Application</string>
</resources>
```

Fig. 5.2.3.3.26 Cambio del nombre de la aplicación en *string.xml*

- A continuación modificamos el nombre de los ficheros principales, tanto el .java como .cpp para posteriormente modificar en cascada todas las referencias a estos que realizamos en el resto de módulos. En definitiva, sustituimos todas las apariciones de la cadena “ImagenTargets” que aparecen en los ficheros con el cuidado de no modificar referencias a otros módulos (“ImagenTargetsRenderer.java”).
- Cambiamos también el nombre del “*DataSet”* para que sea más coherente con el proyecto y las referencias que hacíamos en su carga en el código de él ahora APicture.cpp

- Hecho esto, vamos a cambiar el nombre del paquete referenciado al comienzo de cada módulo para que coincida con la nueva denominación y las respectivas referencias al paquete en la carga de código nativo y otros usos.
- Ahora cambiamos el nombre de las librerías que se identificaban con el antiguo nombre (libImagenTargets.so -> libAPicture.so).
- Por último, hemos de cambiar el código del compilador del proyecto para que haga referencia al nuevo nombre. Para ello, accedemos al fichero "*jni/Android.mk*" y como en los casos anteriores buscamos y sustituimos las apariciones de *"ImagenTargets"* por *"APicture"*

o Cambio de los Iconos.

- Esta parte es la más sencilla de todo el proceso, solo hemos de crear el icono que queramos con cualquier editor gráfico de unas dimensiones iguales a los anteriores iconos y guardarlos en las mismas carpetas (res/drawable...). Hecho esto, solo hemos de acceder a APicture.java y dejar el siguiente código:

```
/** Called when the activity first starts or the user navigates back
 * to an activity. */
protected void onCreate(Bundle savedInstanceState)
{
    DebugLog.LOGD("APicture::onCreate");
    super.onCreate(savedInstanceState);

    // Set the splash screen image to display during initialization:
    mSplashScreenImageResource = R.drawable.apicture_logo;
```

Fig. 5.2.3.3.27 Cambio del icono de inicio.

- Una vez hemos cambiado el de inicio de la aplicación, vamos a cambiar el icono de la propia aplicación, el que nos aparecerá en nuestro terminal para ejecutarla.

```
<application
    android:icon="@drawable/apicture_icon"
    android:label="@string/app_name"
    android:description="@string/app_description"
    android:launchMode="singleTask"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
    android:debuggable="true">
```

Fig. 5.2.3.3.28 Cambio del icono de la aplicación en el terminal.

Ya podemos dar por concluida la codificación de todas las clases necesarias para el funcionamiento de nuestra aplicación.

3. Escritura de la Memoria del proyecto

Ya tenemos el proyecto acabado, tal como habíamos pensado, nuestra aplicación es capaz de reconocer una serie de cuadros y mediante la arquitectura y el sistema de RA nos proporciona una interfaz para recibir información acerca del mismo.

Ahora, nos toca realizar la documentación y la memoria del proyecto. Para la escritura de la memoria se ha optado por el uso de Google-doc que nos permite acceder a la misma desde la nube, esto es, desde cualquier lugar con acceso a internet y sin miedo a poder sufrir pérdidas en la misma debido a diferentes imprevistos tales como caídas de la red eléctrica o sobreescrituras accidentales. Además, fue determinante la posibilidad que nos proporciona esta herramienta de permitir el acceso al documento de los invitados que deseemos, en nuestro caso le damos acceso al tutor del proyecto para que pueda mantener una revisión constante de la evolución de la memoria y añadir notas de corrección. Pese a lo ya expuesto acerca del almacenamiento en la nube, hemos ido realizando copias periódicas en diferentes equipos para mantener copia de seguridad que nos sirvan de respaldo en caso de problemas externos o en caso de no disponer del acceso a internet necesario para acceder a los servidores de Google.

Una vez confirmada la corrección de la memoria, realizamos la tarea de migrar el documento al estándar de ofimática Microsoft Office Word, donde le daremos el formato final al documento e incluiremos los últimos componentes de la memoria, como son el índice del contenido y la portada proporcionada por la Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación**,** a la que pertenece nuestra carrera. Finalmente, ya podemos dar por concluido el desarrollo de nuestro proyecto final de carrera como Ingenieros Técnicos Informáticos de Gestión.

6. Conclusiones

Ha sido la primera vez que yo, personalmente, he realizado un trabajo tan complejo y largo. Por ello, antes de decidirme por este proyecto en particular, consulte con todos los profesores que me fue posible los proyectos que me podrías ofrecer. Tras sopesar todas las posibilidades de las que disponía, decidí realizar un proyecto que me proporcionaste la satisfacción de ver como progresaba y tomaba forma con el paso del tiempo y que al mismo tiempo tuviese una funcionalidad práctica. Por ello es por lo que en un primer momento descarte aquellos proyectos cuya esencia consistirá en la investigación de campos desconocidos hasta el momento, pese a las múltiples ventajas que supondría aumentar el conocimiento en esos temas. Así pues, me decante por algo más pragmático, que me motivara la temática de desarrollo y que al mismo tiempo me supusiera un reto personal.

El desarrollo del proyecto se ha realizado en solitario, a pesar de que con más componentes en el desarrollo podríamos haber llegado a realizar una aplicación más compleja y completa. Esto se realizó de este modo al no conocer la verdadera magnitud de la idea inicial y partir de un punto básico acerca de las tecnologías y herramientas de RA. Sin embargo, creemos que el proyecto ha tenido éxito y hemos conseguido llegar a buen puerto y realizar lo que se buscaba al comienzo del mismo.

Respecto al lenguaje de desarrollo y la plataforma para la que se ha realizado, se trata de la primera aplicación que desarrollo y me ha resultado gratificante conseguir realizarla con éxito. Al partir desde cero en la programación para Android, ha resultado una tarea muy laboriosa el estudio y comprensión de toda la arquitectura del SO y su funcionamiento, desde el uso de los componentes de la interfaz gráfica hasta la comunicación entre Activity's de las aplicaciones. El uso de herramientas como el entorno de desarrollo Eclipse y el conocimiento previo del lenguaje Java ha facilitado la implementación del proyecto y fueron tenidos en cuenta a la hora de la elección del mismo ya que hasta la fecha ha sido el lenguaje de programación más amigable que he usado y con el que sentía predisposición para trabajar en el PFC. Por otro lado, muchas veces la página del API de Android resultó confusa e incompleta y me vi obligado a recurrir a páginas adicionales para asimilar el modo de utilización o trabajo de algunas de las clases y métodos utilizados en nuestra aplicación.

En cuanto a la distribución del tiempo dedicado a cada fase del proyecto, se vio alterado por problemas con la disponibilidad de un terminal en el que comprobar e instalar la aplicación y por lo tanto, se vio alargado más de lo que se tenía planeado en un primer momento. Al comienzo del proyecto, no disponía de un smartphone en el que instalar los ejemplos y tutoriales de aprendizaje y me vi obligado a trabajar con el simulador del SDK de Android, que si bien resulta muy útil y suficiente al comienzo, posteriormente se pudo comprobar que era necesario disponer de un smartphone. No fue hasta pasados unos meses cuando se pudo obtener, debido a problemas con la compañía telefónica, el terminal necesario y entonces es cuando el proyecto puedo obtener el ritmo necesario para su resolución en los tiempos planteados. En cuanto a

la distribución y balance de las diferentes etapas, queda claro que a la fase de análisis de los requisitos del proyecto ha de ser la más crítica al suponer cualquier fallo en esta, una cambio en cadena del resto de fases; por lo que dedicamos a ella todo el tiempo necesario para que, al pasar de fase, estemos seguro de la precisión con la que hemos descrito el problema. Sin embargo, la fase de implementación del código resultó la más extensa debido sobre todo a que hubo código que estuvo en continuo cambio y en cada iteración tuve que reescribirlo desde 0 para su mejor funcionamiento.

Aun con todo esto, teniendo en cuenta mis conocimientos iníciales acerca del lenguaje de programación, la programación para smartphones, el desconocimiento del campo de la Realidad Aumentada de los implicados en el proyecto y la novedad de muchas de las herramientas utilizadas; considero todo un logro haber conseguido realizar el proyecto en curso, hemos conseguido alcanzar todos los objetivos perseguidos en un tiempo de desarrollo óptimo y con una calidad notable para tratarse de la primer toma de contacto con todo lo desarrollado.

7. Líneas Futuras

El resultado del proyecto no ha sido más que una pequeña muestra de lo que podemos realizar con el sistema creado durante el mismo, estamos trabajando sobre 2 cuadros de Velázquez y la información mostrada ha sido recopilada de páginas de interés general.

Por un lado, con el sistema ya creado, podríamos añadir tantos cuadros sobre cualquier autor que fuese y extender de este modo su funcionalidad tanto como deseásemos. Esto nos podría llevar a plantearnos, con mayores recursos monetarios, la transformación del sistema para sustituir el trabajo con BBDD locales, en el terminal, por base de datos remotas a donde nos conectaríamos en busca de la información en cuestión. Para ello, habría que contratar servidores y demás infraestructura que, debido a que se trata de un proyecto universitario, no disponemos actualmente. Este cambio no sólo nos permitiría ampliar la frontera del número de cuadros de los que seríamos capaces de proporcionar información, sino que además aligerará de una manera importante el peso o tamaño de la aplicación al almacenar no sólo las bases de datos con la información relativa a los cuadros, sino también toda las imágenes utilizadas en la misma, que suponen un peso importante de la aplicación a la hora de su instalación.

Por otro lado, la información mostrada puede resultar escasa, imprecisa o incluso, dependiendo de la fuente de la misma, irreal. Esto, como se ha comentado, es consecuencia del hecho de haberse recopilado de páginas de interés general y no haber profundizado en su investigación. Así, si añadiéramos a algún experto en historia del arte, podríamos ampliar, mejorar y corregir la información suministrada al usuario en la aplicación de forma que le resulte de mayor fiabilidad y confianza.

Además, en el proyecto estamos trabajando con la idea y el objetivo de aplicar la tecnología de la realidad aumentada sobre cuadros pictóricos famosos y con un trasfondo ampliamente estudiado durante sus años de historia. Sin embargo, este mismo sistema nos podría permitir interaccionar de igual modo con objetos tales como menús de restaurantes que al pulsar sobre las fotos nos mostrará la información acerca del plato, podría servirnos para obtener información acerca de una sencilla fachada arquitectónica o incluso podría darnos aquello que deseemos conocer acerca de un libro partiendo de su portada.

Por último, con un mayor conocimiento de programación del lenguaje, podríamos mejorar el rendimiento y eficacia de la aplicación para corregir algunos defectos de “lag” o retardo en la respuesta, que se producen en determinados momentos de la aplicación. A todo ello, podríamos sumarle una representación tridimensional de los objetos de interés que podríamos realizar si tuviésemos los conocimientos de Open GL necesarios, mostrando una interfaz más vistosa y llamativa que hiciera más atractiva nuestra aplicación de cara al usuario, pero que sin embargo no añadiría funcionalidades.

8. Bibliografía

1. Realidad Aumentada

- REVISTA ICONO 14, 2011, Año 9 Vol. 2, pp. 212-226. ISSN 1697-8293. Madrid (España), David Ruiz Torres: Realidad Aumentada, educación y museos.
- http://www.inglobetechnologies.com/docs/whitepapers/AR_editoria_whitepaper_es.pdf
- http://www.anobium.es/docs/gc_fichas/doc/6CFJNSalrt.pdf

2. JDK

- http://www.lab.dit.upm.es/~lprg/entorno/mipc/jdk/index.html
- http://www.usmp.edu.pe/publicaciones/boletin/fia/info49/too.html
- http://www.oracle.com/technetwork/java/javase/downloads/index.html

3. Herramientas Android

- http://developer.android.com/intl/es/sdk/index.html
- http://es.wikipedia.org/wiki/Desarrollo_de_Programas_para_Android
- http://zsoluciones.com/datos/?p=246

4. Eclipse

- http://es.wikipedia.org/wiki/Eclipse_(software)
- http://www.eclipse.org/
- http://es.slideshare.net/Benedeti/ide-eclipse-breve-gua-201399

5. Cygwin

- http://www.cygwin.com/
- http://es.wikipedia.org/wiki/Cygwin
- http://ubuntulife.wordpress.com/2008/10/22/cygwin-un-entorno-de-linux-para-tu-windows/

6. Qualcomm y Vuforia

- https://developer.qualcomm.com/mobile-development/mobile-technologies/augmented-reality
- http://www.qualcomm.com/about/history

7. API Android

- http://developer.android.com/intl/es/reference/packages.html

8. Aprendizaje de Android

- http://www.android-spa.com/
- http://www.edu4java.com/android.html
- http://www.javahispano.org
- http://stackoverflow.com